Pedro David Netto Silveira

Avaliação e Extensão de uma Ferramenta de Monitoramento de Sistemas Computacionais em Rede: Sistema de Diagnóstico Instantâneo

Vitória, ES
2009

Pedro David Netto Silveira

Avaliação e Extensão de uma Ferramenta de Monitoramento de Sistemas Computacionais em Rede: Sistema de Diagnóstico Instantâneo

Monografia apresentada para obtenção do grau de Bacharel em Ciência da Computação pela Universidade Federal do Espírito Santo.

Orientador:
Prof.ª Dr.ª Roberta Lima Gomes
Co-Orientador:
Prof. Dr. Magnos Martinello

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA

Vitória, ES
2009

Pedro David Netto Silveira

Avaliação e Extensão de uma Ferramenta de Monitoramento de Sistemas Computacionais em Rede: Sistema de Diagnóstico Instantâneo

Banca Examinadora:

_____

Prof.ª D.Sc. Roberta Lima Gomes
Orientador

_____

Prof. D.Sc. Magnos Martinello
Co-Orientador

_____

Prof. M.Sc. Jadir Eduardo Sousa Lucas
Examinador

Vitória, _____ de _____ de _____.

DEDICATÓRIA

Dedico este trabalho ao meu pai, que sempre me apontou a direção correta impedindo meus tropeços.

AGRADECIMENTOS

Em primeiro lugar, à Deus, que nunca me desamparou.

À minha mãe, Márcia e meu pai, Iraldo, pela fé que depositaram em mim, e por sempre estarem presentes nos momentos mais difíceis, dando todo amor que pudessem dar, e à minha querida irmã, Carol pelo carinho, conversas animadoras e por sua alegria contagiante.

A Marcellus e Juliana (casal favorito!) e aos queridos Kelly, Carlão e Snarf. Aprendi a amá-los como irmãos. A ajuda de vocês foi essencial para essa conquista.

À minha noiva, Flavinha, por estar sempre presente, me ajudando, me incentivando e sendo companheira.

Ao Núcleo de Cidadania Digital, por me fazer o profissional que sou. Abraço especial ao amigo Beto.

Aos meus orientadores, Roberta e Magnos, por toda a disposição com relação a este trabalho, e pelo incentivo.

Ao Pessoal do PrD, especialmente, ao Bruno, Vinicius e Diego, pelo auxilio e prontidão nas respostas, e principalmente pelo cuidado e atenção que deram às minhas dúvidas.

“Mas em todas estas coisas somos mais do que vencedores, por aquele que nos amou.”

Romanos 8:37

RESUMO

O contínuo crescimento das redes de computadores, e a forte expansão da Internet, demandam ferramentas que realizem a gerência e o monitoramento geral dos sistemas computacionais na intenção de estabilizar as situações emergenciais. O avanço da tecnologia em busca dessas ferramentas incita o desenvolvimento das mais variadas formas de fazer monitoramento. Um projeto que nasceu na Universidade Federal do Paraná, denominado Sistema de Diagnóstico Instantâneo (SDI), observando este fato, procurou apresentar uma abordagem contemporânea, que enfatiza e dá valor a liberdade que um sistema pode proporcionar. Essa monografia inclui implantar o SDI no contexto de um laboratório computacional, promovendo um estudo que visa apresentar extensões que adaptassem o sistema às necessidades do laboratório e que também pudessem de alguma forma contribuir para o projeto SDI.

Palavras-chave: SDI, Gerenciamento, SSH, Portabilidade, Integração.

LISTA DE SIGLAS/ACRÔNIMOS

AIX - Advanced Interactive eXecutive

API - Application Programming Interface

BSD - Berkeley Software Distribution

CACIC - Configurador Automático e Coletor de Informações Computacionais

CPU - Central Processing Unit

DI - Departamento de Informática

DLL - Dynamic-*link* library

DOS - Disk Operating System

FISL - Feira Internacional do Software Livre

GB - Giga Byte

GHz - Giga Hertz

GPL - General Public License

HD - Hard Disk

HP-UX - Hewlett Packard-UniX

HTTP - Hypertext Transfer Protocol

HTTPS - Hypertext Transfer Protocol Secure

ICMP - Internet Control Message Protocol

IP - Internet Protocol

IPSO - Internet Protocol for Smart *Object*s

LPRM - Laboratório de Pesquisa em Redes e Multimídia

NNTP - Network News Transfer Protocol

MB - Mega Byte

Mbps - Mega bits por segundo

OCS - Open Computer and Software

OCSI - Open Computer and Software Inventory

PaQueT - Packet Query Tool

PHP - HyperText Pre*processor*

po - *parser object*

POP3 - Post Office Protocol

PRD - Paraná Digital

RAID - Redundant Array of Independent Drives

RAM - Randomic Acess Memory

RRDTool - Round Robin Database Tool

SGDB - Sistema Gestor de Base de Dados

SMTP - Simple Mail Transfer Protocol

SNMP - Simple Network Management Protocol

SQL - Structured Query Language

SSL - Secure Sockets Layer

SSH - Secure SHell

sshd - secure shell daemon

TCP - Transmission Control Protocol

UDP - User Datagram Protocol

UFES - Universidade Federal do Espírito Santo

UFPR - Universidade Federal do Paraná

VPN - Virtual Private Network

WAN - Wide Area Network

WWW - Word Wide Web

XML - eXtensible Markup Language

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Com a crescente popularização da Internet, as redes de computadores vêm assumindo um papel cada vez mais importante dentro do contexto computacional. Hoje parece ser impossível conceber um modelo de negócios, seja no setor empresarial, acadêmico ou industrial, sem a existência da Internet, do computador, e das redes às quais estão conectados. Além disso, existem os serviços oferecidos, que possibilitam a comunicação e seu funcionamento geral.

Então como saber se o serviço de correio eletrônico está operacional? Se o roteador de uma rede falhar, em qual momento o administrador será avisado? Um computador teve o processador danificado, o que ocasionou isso? Em qual momento do dia ocorre um maior consumo de banda? Algum usuário apresenta consumo maior de banda?

Essas perguntas mostram que somente oferecer recursos e serviços em uma rede não é suficiente, também é necessário que ela seja rápida, segura e disponível. A função do gerenciamento dos sistemas computacionais em rede é justamente fazer com que esses objetivos sejam alcançados.

Segundo Black (2008), o gerenciamento permite controle sobre os recursos da rede, assim como a identificação e prevenção de problemas, sendo tal investimento justificado quando se quer o controle dos recursos, de sua complexidade, serviços melhores e controle de custos. O gerenciamento envolve basicamente cinco objetivos: desempenho, segurança, proteção contra falhas, configuração e contabilização.

Existe uma ferramenta de gerência desenvolvida no meio acadêmico, chamada Sistema de Diagnóstico Instantâneo (SDI), que permite administrar, coletar informações e gerar estatísticas de qualquer computador monitorado. No SDI, além dessas funções, há ainda um propósito mais específico: gerar alarmes

quando uma falha acontecer. Nele também é permitido executar processos em qualquer computador monitorado com um ambiente de execução distribuído.

1.1 OBJETIVO

O principal objetivo deste trabalho foi estudar e implantar o SDI no Laboratório de Pesquisas em Redes e Multimídia (LPRM). Além da implantação deste *software*, outro objetivo apresentado por esta proposta de trabalho, envolve a realização e avaliação de extensões no sistema visando adequar o SDI ao contexto do LPRM. Isso só é possível, pois o código fonte do sistema está disponível na Internet para alterações e redistribuição.

De uma forma geral, a proposta de extensão envolve *(i)* promover uma portabilidade para o SDI que atualmente funciona somente entre sistemas operacionais Linux, *(ii)* verificar a possibilidade de integração entre o SDI com outros sistemas de mercado que apresentem uma proposta de funcionamento similar ao SDI, e existindo essa possibilidade, realizá-la, e por último, *(iii)* fazer com que o SDI além de detectar as falhas durante o monitoramento também seja capaz de resolvê-las.

1.2 METODOLOGIA

Inicialmente, foi realizado um estudo sobre o funcionamento do SDI, baseado na documentação existente na ferramenta *wiki* disponibilizada pelos desenvolvedores na Internet. O estudo também foi realizado com testes empíricos e avaliação de código fonte. Paralelamente, também foram realizados outros estudos de sistemas similares na intenção de fazer uma avaliação comparativa com o SDI.

Após ter adquirido o conhecimento básico, foi realizada a instalação do sistema em um dos servidores de teste do LPRM. Feita a instalação, o sistema foi

testado com a configuração padrão. Posteriormente novas configurações foram adicionadas visando um aprofundamento do entendimento do sistema.

Assim que o entendimento sobre o funcionamento do sistema já estava bem maduro, a ponto de entender passo a passo como ele funciona, deu-se inicio à fase de realização das propostas de extensão, primeiro desenvolvendo a portabilidade, depois a integração com outras ferramentas, e por último o desenvolvimento de *scripts* que não somente iriam monitorar, mas também teriam a função de corrigir problemas.

1.3 ESTRUTURA DA MONOGRAFIA

Esta monografia apresenta em seus primeiros capítulos uma abordagem teórica sobre o funcionamento geral do SDI, que servem de base para os seguintes, que apresentam as atividades realizadas durante este trabalho.

No capítulo 2 são apresentados alguns sistemas que de alguma forma possuem um funcionamento semelhante ao SDI.

O capitulo 3 aborda de maneira bastante abrangente o SDI, incluindo seus objetivos, funcionamento, arquitetura e exemplos de atuação dos *scripts* usados pelo sistema na rede.

No capítulo 4 são apresentadas as propostas de extensão que foram realizadas e as contribuições que esta monografia gerou para o projeto que mantém o SDI.

O capítulo 5 mostra os resultados avaliativos das extensões, principalmente com relação à portabilidade.

Finalmente o capítulo 6 apresenta a conclusão deste trabalho, e sugestões para trabalhos futuros.

2 TRABALHOS CORRELATOS

Neste capítulo serão abordados alguns sistemas atuais que, de certa forma semelhante ao SDI, são empregados no monitoramento de sistemas computacionais em rede. Visa-se com isso realizar um levantamento das principais características desses sistemas, comparando-os com o SDI.

2.1 MONITORAMENTO

Segundo Sakellariou (2004), monitoramento é o ato de coletar informações sobre as características e estado dos recursos da rede, os quais o administrador tem algum interesse.

> O gerenciamento de rede inclui o oferecimento, a integração e a coordenação de elementos de hardware, software e humanos, para monitorar, testar, consultar, configurar, analisar, avaliar e controlar os recursos da rede, e de elementos, para satisfazer à exigências operacionais, de desempenho e de qualidade de serviço em tempo real a um custo razoável. (KUROSE, 2006, pg. 575)

De acordo com Kurose (2006), a infa-estrutura do gerenciamento da rede engloba três componentes principais. São eles, a entidade gerenciadora, o dispositivo gerenciado, e o protocolo de gerenciamento a ser usado.

A entidade gerenciadora pode ser qualquer uma das aplicações que serão comentadas até o final desse capítulo. Em geral são as grandes responsáveis pela coleta, o processamento a análise e a apresentação das informações obtidas.

O dispositivo gerenciado, em geral é um equipamento de rede. Esse dispositivo poderia ser um *host*, um roteador, um *hub*, impressora, um *software*, ou até mesmo um sistema computacional.

O terceiro componente tem sua execução efetuada entre a entidade gerenciadora, e o agente de gerenciamento da rede, que é um processo executado no dispositivo gerenciado a fim de realizar as ações locais, sob o comando da entidade gerenciadora.

Os agentes somente podem informar à entidade gerenciadora sobre a ocorrência de um evento excepcional, pois o protocolo de gerenciamento efetua a comunicação entre as partes envolvidas.

2.1.1 Tipos de Ferramentas de Monitoramento.

Segundo Moceri (2004), as ferramentas de monitoramento de desempenho da rede variam de uma simples aplicação até um sistema hierárquico complexo. Alguns sistemas usam monitoramento passivo para obter informações sobre o tráfico da rede enquanto outros ativamente realizam *polling* nos dispositivos da rede para obter as informações de desempenho. A seguir, as classes de monitoramento, com abordagens e ferramentas diferentes para cada caso.

a) Monitoramento Integrado de Plataformas SNMP

São Sistemas que usam SNMP (Simple Network Management Protocol) para dar uma visão completa da rede. As Plataformas de monitoramento trabalham ativamente coletando informações sobre os dispositivos da rede e analisam os dados. Os administradores de rede têm a flexibilidade para decidir quais os parâmetros que deseja monitorar e como as informações serão reportadas.

Esses sistemas de monitoramento fornecem uma análise detalhada e completa do desempenho da rede. Características de desempenho comuns incluem a utilização de banda, *throughput*, tempo de resposta e as taxas de erro, bem como carga da CPU e utilização de memória do equipamento de rede e servidores.

b) *Passive Analysis*

São ferramentas passivas que medem desempenho de rede, também chamadas de ferramentas de captura de pacotes, ou *sniffers*. Podem ser conceituadas como a classe de ferramentas que não geram tráfego na rede enquanto coletam os dados, ao invés disso, o analisador, apenas permanece escutando o tráfego que acontece na rede. Devido a esse fato, pode-se dizer que a amplitude da análise gerada é limitada, tendo em vista, que a ferramenta passiva, apenas pode contemplar o tráfego local, onde se encontra o dispositivo que executa o analisador.

c) Monitoramento de Aplicativos e Serviços

Refere-se à classe de ferramentas que provê monitoramento de aplicações individuais de rede. O monitoramento de aplicações depende menos do equipamento e infra-estrutura de redes e mais dos servidores que irão prover serviços aos usuários.

Um uso comum dessas ferramentas é o monitoramento de disponibilidade de sites. Entretanto, esse monitoramento pode ser realizado em cima de protocolos bem mais complexos que o HTTP, como por exemplo, os protocolos que são usados para o monitoramento de bancos de dados.

d) Monitoramento de Fluxo

Ao invés de olhar para o tráfego no nível de pacotes, esse tipo de monitoramento analisa o tráfego que ocorre na rede como fluxo. O monitoramento de fluxo visualiza o tráfego no nível de conexões individuais, usuários, protocolos, ou aplicações. Isto permite que ferramentas de monitoramento de fluxo tenham uma visão mais ampla sobre a rede, incluindo informação específica sobre a aplicação, a performance da conexão e até mesmo sobre a segurança da rede.

2.1.2 Comparação dos Tipos de Ferramentas de Monitoramento.

A tabela 2-1 mostra os diferentes tipos de ferramentas de monitoramento mencionados, e destaca as principais diferenças entre elas.

Tabela 2-1: Comparação dos diferentes tipos de ferramentas de monitoramento

Tipo	Passivo ou Ativo	Camada de operação	Operação Básica
Monitoramento Integrado de Plataformas SNMP	Ativo	Enlace e Rede	Uma plataforma de gerenciamento que elege agentes nos dispositivos que irão obter informações sobre a rede.
Passive Analysis	Passivo	Enlace	Ouve e captura pacotes que trafegam pelos dispositivos da rede para análise.
Monitoramento de Aplicativos e Serviços	Ativo	Aplicação	Monitoramento ativo de servidores a fim de obter informações de desempenho.
Monitoramento de fluxo	Ativo	Transporte	Monitora as conexões e fluxo na rede nas camadas mais altas.

Fonte: <http://www.cs.wustl.edu/~jain/cse567-06/ftp/net_traffic_monitors2.pdf>, acesso: 30/11/2009

A linha do monitoramento de redes e seus recursos é uma linha que hoje é bastante explorada, visto que é impossível manter o funcionamento ideal de grandes ou pequenas redes de computadores, se não houver uma ferramenta responsável por gerar os alarmes quando uma situação crítica for atingida. Além disso, é de suma importância controlar o conteúdo que trafega dentro dos roteadores internos, e gerar estatísticas que irão dizer o que vem acontecendo na rede durante os últimos tempos.

A opção do monitoramento da rede, que inclui monitoramento dos protocolos, das conexões ou controle de banda pode até ser aplicada ao SDI, que foi o aplicativo usado para o desenvolvimento desse trabalho, mas não é nativamente o objetivo do sistema.

Serão apresentados em seguida alguns sistemas de monitoramento, que assim como o SDI, permitem o diagnóstico sobre o estado geral de uma rede de computadores, ou dos recursos dessa rede.

2.2 PANDORA FMS

Pandora FMS (*Free monitoring system*)[1] é um aplicativo de monitoramento desenvolvido pela "Ártica Soluciones Tecnológicas" com o intuito de assistir sistemas e aplicações.

De maneira geral, esse *software* pode monitorar *hardware*, *software* além do próprio sistema operacional. Os avisos que esse monitoramento gera, inclui a exibição dos dados na interface, envio de *e-mails* e até mesmo envio de SMS's.

Pandora é uma ferramenta *OpenSource* e pode ser implementado sobre muitos SO's, com agentes específicos para cada plataforma. Já existem agentes para o Windows, Linux, Solaris, HP-UX, BSD, AIX e IPSO.

Segundo Urrea (2009), o Pandora não visa simplesmente informar se existe algum parâmetro certo ou errado, o sistema quantifica um estado, podendo armazenar um valor numérico (em uma escala) referente à uma situação. Dessa forma, se houver algum valor que ultrapasse os limiares impostos pelo sistema, um alarme é gerado.

O sistema possui um banco de dados, para que possam ser gerados relatórios, estatísticas e mensurações sobre aplicações e sistemas de *hardware* como: *firewalls*, *proxies*, servidores *Web*, VPN, roteadores, *switches*, processos, serviços, *etc*, tudo integrado em uma arquitetura aberta e distribuída.

2.2.1 Arquitetura

O esquema de funcionamento do Pandora passa por 3 componentes principais:

- *Pandora FMS Server*: Possui quatro diferentes tipos de servidores:
    1. Core Server: É o receptor de pacotes de dados gerados pelos agentes, e tem a função de processar esses dados.

[1] Site oficial: http://pandorafms.org/

2. *Network Server*: Monitoriza os sistemas remotos utilizando recursos como: ICMP, TCP, UDP ou consultas SNMP. Esses servidores agem sozinhos, como agentes da rede e reúnem as informações remotamente
3. *Recon Server*: Varre a rede detectando novos sistemas, quando encontrados, são adicionados ao Pandora para que possam ser monitorados também.
4. *SNMP console*: Recebe e processa SNMP e define alertas associados.

- Banco de dados: Contem as informações salvas, necessárias para o trabalho do Pandora.
- Agentes: Eles são executados em cada sistema local, e foram desenvolvidos para trabalhar em uma plataforma específica, utilizando ferramentas específicas de cada sistema que foi acolhido.

A arquitetura do Pandora é bem descentralizada,tendo como componente vital a base de dados (atualmente só suporta MySQL) onde tudo é armazenado. Além disso, existem os servidores que coletam (através dos agentes) e processam os dados e posteriormente os introduzem na base de dados. O console se encarregará de mostrar os dados presentes na base de dados.

Por fim, o Pandora é uma ótima opção para o monitoramento de rede e seus serviços, disponibilizando uma visão gráfica e consistente das informações, mas com relação ao monitoramento de recursos, não é aconselhável, pois esse não é o seu objetivo primordial, assim como o SDI. Pandora é uma ferramenta que será a alternativa ideal para as redes de grande porte no futuro.

2.3 OCS INVENTORY

Open Computer and Software Inventory[2] é uma aplicação desenhada para ajudar um administrador de rede ou sistemas acompanhar a configuração do computador e do *software* instalado na rede.

OCS Inventory é um *software* GPL, ou seja, livre para usar e copiar. Ele também é Open Source, (é possível modificar o código fonte do programa, no entanto, para atualizar o código e redistribuí-lo, deve-se fornecer as atualizações nos termos da licença GPL).

No OCSI, o diálogo entre cliente e servidor, é baseado nos padrões HTTP/HTTPS, e usa-se XML para a formatação dos dados. Com isso, o sistema promove um inventário *online* de todos os dispositivos acoplados a uma rede de computadores, fazendo análise geral de tanto *hardware* como *software*.

2.3.1 Arquitetura

O serviço de gerenciamento possui quatro componentes principais:

- *Database Server*: Responsável por armazenar as informações de inventário.
- *Communication Server*: Responsável por manipular as comunicações HTTP entre o *database Server* e os agentes.
- *Administration Console*: Permite os administradores a buscar a informação no banco de dados por meio do navegador.
- *Deployment Server*: Responsável por armazenar as configurações requeridas pelos pacotes implementados.

Comunication server é escrito em *Perl* como um modulo do apache (*Web Server*), dessa forma, os *scripts* que também são escritos em Perl, são executados, quando o apache iniciar. O *comunication Server* será responsável

[2] Site Oficial: http://www.ocsinventory-ng.org/

por levar as informações coletadas pelos agentes manipulados pelo Deployment Server até o *database server* que atualmente suporta apenas MySQL. O *console* é escrito em PHP é funciona como a interface entre o administrador e o sistema.

Finalmente, o *OCS Inventory* é uma alternativa muito parecida com o SDI, e tem como função principal o gerenciamento dos recursos dos computadores de uma rede. É bastante usada para redes de pequeno e médio porte, é a desvantagem desse sistema em relação ao SDI, se relaciona com a liberdade proporcionada pelo SDI para a execução de tarefas específicas.

2.4 CACIC

O Configurador Automático e Coletor de Informações Computacionais (CACIC)[3] trata-se de uma solução de inventário de *hardware* e *software*. Foi lançado no 6º Fórum Internacional de *Software* Livre (FISL) pela Dataprev (Empresa de Tecnologia e Informações da Previdência Social).

Como dito por Freitas (2008), a princípio, a idéia do CACIC, era atender as demandas internas do governo, mas com o tempo, o sistema foi adquirindo fama a um ponto de extrapolar o setor público federal. Isso mostrou que o *software* poderia atender até mesmo à sociedade. Tanto é que em pouco tempo, após a liberação do *software*, formou-se uma extensa rede de usuários e desenvolvedores.

O CACIC pode fornecer um diagnóstico preciso dos *hosts*, informar sobre os *softwares* utilizados e licenciados, informar sobre configurações de *hardware etc*. As principais funções do CACIC englobam a coleta e disponibilização de informações sobre *software* e *hardware* de uma maneira geral, gerar alarmes para os administradores do sistema quando situações de risco são detectadas, providência de informação necessária para tomadas de ações imediatas.

[3] Site oficial: http://www.serpro.gov.br/servicos/downloads/cacic

2.4.1 Arquitetura

Segundo Kreuch (2007), o CACIC é divido em dois componentes principais, o módulo gerente e o módulo agente, sendo que o módulo gerente está localizado em qualquer plataforma Linux que possua um Apache (*Web Server*), MySQL (Banco de dados) e PHP (linguagem de *script*). O módulo gerente pode ser manipulado pela interface *Web* provida pelo sistema. A partir daí, é possível realizar a administração de todos os agentes.

O módulo agente apenas é suportado por dois tipos de SO, o Windows e o Linux, sendo que o agente feito para Windows foi desenvolvido *Delphi* (usando apenas as bibliotecas livres) e o de Linux em *Perl* e *Python.* A função básica deste módulo é capturar as informações dos *hosts* e enviá-las para o gerente.

Por fim, o CACIC foi abordado neste estudo, por ser um software brasileiro, assim como o SDI. Seu objetivo é bastante parecido com o SDI e OCSI, o monitoramento dos recursos dos computadores da rede, e apresenta mesma desvantagem que o OCSI, não provê a mesma liberdade de criação de *scripts* específicos, como o SDI.

2.5 PAQUET

Essa ferramenta foi desenvolvida por Natasha Petry Ligocki como uma extensão do Sistema Gerenciador de *Streams* de Dados Borealis, fruto de sua dissertação de mestrado. De acordo com Ligocki (2008), o PaQueT (Packet Query Tool) é uma ferramenta de propósito geral, que permite o administrador de redes definir consultas de acordo com as necessidades específicas que sua rede possui.

O sistema trata todos os pacotes de uma rede, modificando-os de acordo com um esquema pré-definido e direciona a informação para o Sistema Gerenciador de Banco de Dados (SGBD). O administrador agora pode consultar a base de dados usando o SGBD para fazer as consultas que lhe convém.

O PaQueT foi projetado para atender um grande de necessidades de monitoramento. O administrador pode definir qual o tipo de informação ele deseja obter em um processamento parecido com a realização de uma consulta em um banco de dados.

> O PaQueT permite que se monitorem valores desde métricas no nível de pacotes àquelas normalmente fornecidas apenas por ferramentas baseadas em *Netflow* e SNMP. Além disso, a ferramenta possui uma curva de aprendizagem acentuada, permitindo que se obtenha a informação desejada rapidamente. (Ligocki, 2008, pg V).

2.5.1 Arquitetura

A ferramenta possui um conjunto de esquemas pré-definidos chamado de *Packet Schema,* os quais descrevem a estrutura dos pacotes. O Paquet consiste em basicamente 3 módulos:

- *SGBD Borealis*: Requer que o esquema de dados (*Packet Schema*) estejam previamente definidos para que eles possam ser processados.
- *IP Tool*: Responsável pela coleta de pacotes e de sua decomposição nos campos previamente definidos no *Packet Schema*.
- *Dynamic Stream*: Recebe um documento XML como entrada, que contém as definições de consultas e tem a função de registrar a consulta no SGBD.

Assim como qualquer outro SGBD, o Borealis requer que os dados de entrada sejam previamente definidos em campos definidos em campos, antes de realizar a inserção. O PaQueT possui um conjunto de esquemas pré-definidos, que descrevem a estrutura dos pacotes que trafegam pela rede. A função do IPTool, é justamente após a capturar e decompor os pacotes em campos, de acordo com o esquema. Esses dados já decompostos serão inseridos no banco de dados pelo Borealis, onde poderão ser efetuadas as consultas.

O PaQueT, assim como o SDI é um sistema desenvolvido no meio acadêmico, porém o Paquete foi desenvolvido com o intuito maior de monitoramento dos serviços de rede e protocolos, não para recursos computacionais ou inventário.

Além disso, os relatórios gerados são impressos em arquivos de *log*, ou arquivos texto, não em uma interface gráfica.

2.6 CACTI

Cacti[4] é uma ferramenta *freeware*, muito usada no mercado para administração geral sobre os estados de uma rede de computadores. O Cacti hoje se encontra na versão 0.8.7, que permite uma visão gráfica das informações coletadas. Conforme Black (2008) por trás do Cacti, existe uma ferramenta *RRDTool*, que realiza todo o trabalho, armazenando os dados necessários para a criação dos gráficos, e inserindo esses dados no banco de dados.

> RRDTool é um sistema de base de dados Round-Robin criado por Tobias Oetiker sob licença GPL. Foi desenvolvido para armazenar séries de dados numéricos sobre o estado de redes de computadores, porém pode ser empregado no armazenamento de qualquer outra sére de dados como temperatura, uso de CPU *etc*. RRD se refere a Round Robin Database. (BLACK, 2008,pg. 30)

Esse sistema é robusto e de fácil uso. Foi desenvolvido para se adaptar fácilmente às diversas necessidades, como por exemplo monitorar os estados de todos os elementos da rede e *software*s. Além disso, também pode monitorar largura e banda e os consumos em geral dos componentes de um computador.

De acordo com Black (2008), com o Cacti é possível gerar gráficos referentes a uso de memória física, memória virtual, quantidade de processos, processamento, tráfego de rede *etc*. O que possibilita uma visão mais ampla sobre os relatórios gerados em cima funcionamento da rede e seus recursos.

O Cacti é uma boa opção para o gerenciamento dos recursos da rede, e seu objetivo é muito parecido com o objetivo do SDI, porém o Cacti não proporciona a liberdade aos administradores de prepararem seus próprios *scripts*, diferentemente do SDI.

[4] Site Oficial: http://www.cacti.net/

2.7 NAGIOS

O sistema Nagios[5] é uma aplicação de monitoração de rede, de código fonte aberto, que foi desenvolvido por Ethan Gaustad. Hoje o sistema é mantido por ele em conjunto com muitos outros desenvolvedores, e encontra-se em sua versão, estável, 3.1.0.

De acordo com Barth (2006), a idéia do programa parte do pressuposto de que, quanto mais rápido uma situação crítica for detectada, mais eficiente será o programa, sendo que o conceito de “crítico”, é determinado pelo administrador.

Um conceito importante que existe no Nagios, é o conceito de cores: Verde para serviços cuja atuação está dentro da normalidade, amarelo para situações questionáveis e vermelho para crítico.

Nagios oferece a opção de mostrar os dados graficamente, para uma melhor visualização dos dados, como por exemplo, a carga de uma interface WAN ou uma CPU durante um dia inteiro. O sistema também monitora variados serviços de rede, como: SMTP, POP3, HTTP, SNMP *etc*. Alem desses serviços, o Nagios também monitora recursos computacionais e equipamentos de rede.

Além das possibilidades de monitoramento, o Nagios também pode ser usado para outros fins, como checagem de disco, ou definição hierárquica de rede, e a geração de alarmes é definida pelo usuário, que escolherá a forma de recebimento da notificação: por *e-mail*, SMS, Pager, ou qualquer outra forma definida por *plugin*.

2.7.1 Arquitetura

Segundo Leme (2006), a arquitetura do gerenciamento é definida em 4 partes:

- Gerente: É quem realiza o processamento das solicitações de requisições feitas.

[5] Site Oficial: http://www.nagios.org/

- Agente: É o programa que coleta os dados e os envia para o gerente processar.
- MIB: Consiste na base de informações gerenciáveis, sendo que os recursos gerenciáveis são definidos como objetos.
- Protocolo de gerenciamento de redes: O protocolo padrão é o SNMP.

Conforme Andrade (2006), o Nagios foi construindo em cima de uma arquitetura servidor/agentes, e em uma rede, executa a partir do servidor, os agentes que irão coletar os dados nos hosts e os enviarão para o processamento das informações no gerente, que finalmente se encarregará de incluir as informações na base de dados. Tendo a base de dados atualizada, a informação estará disponível para ser mostrada na interface gráfica do Nagios.

O Nagios é uma forte ferramenta para o monitoramento das redes de computadores. Com ela é possível monitorar os recursos, manter um inventário, e ainda é possível monitorar o ambiente de rede e seus protocolos. De uma forma parecida com SDI, o Nagios oferece liberdade para o administrador preparar seus próprios *scripts* para a execução nos hosts, porém, os *scripts* precisam ser implementados usando um grupo de linguagens de programação específicas e seguir o padrão de resposta que o Nagios exige.

2.8 CONSIDERAÇÕES SOBRE O CAPÍTULO

Neste capítulo foram abordados alguns sistemas de monitoramento de redes e *softwares* que de alguma forma se assemelham ao SDI, no funcionamento, ou no objetivo final da proposta.

Os dois últimos sistemas citados foram implementados e estão em funcionamento no mesmo ambiente computacional que o objeto de estudo deste trabalho, o SDI.

Dentre os sistemas, foi falado sobre o CACIC, que assim como o SDI, também é um *software* brasileiro. Fora do âmbito técnico, pode-se dizer que a grande

diferença entre o SDI e o CACIC, é que o segundo não foi desenvolvido em um ambiente acadêmico como o primeiro.

Na área de projetos acadêmicos brasileiros, a atenção que gira em torno do objeto é menor, como é o caso do PaQueT, que é uma ferramenta muito útil no controle das informações dos pacotes que trafegam na rede. Mesmo com grande potencial, essa ferramenta não é tão difundida como as ferramentas de mercado, devido a não existência de um plano de divulgação estabelecido. A tabela 2-2 mostra todos os sistemas mencionados neste capítulo, e suas principais funcionalidades.

Tabela 2-2: Sistemas similares ao SDI e suas funcionalidades.

Sistema	Operações
Pandora	a. Identifica automaticamente novos sistemas na rede. b. Disponibiliza de teste de desempenho. c. Gera relatórios e gráficos em tempo real. d. A visão gráfica é definida pelo usuário. e. Armazena dados do mês, para desenvolvimento relatórios.
OCS Inventory	a. Apresenta informação de inventário de Hardware e Software. b. Apresenta baixo consumo de banda. c. Apresenta um WebService próprio. d. Apresenta suporte à *plugins* através de uma API
CACIC	a. Identifica automaticamente alterações no *hardware*/*software* dos *hosts.* b. Mantém um inventário dos recursos físicos das máquinas. c. Possibilita a identificação dos compartilhamentos existentes na rede. d. Realiza controle de patrimônios (tombamento). e. Provê informações para que sejam realizadas ações proativas.
PaQueT	a. Efetua monitoramento tanto no nível de pacotes, como no nível de ferramentas baseadas em NetFlow ou SNMP. b. Gera dados em tempo real, minimizando o volume de dados a ser armazenado. c. Estendível para oferecer suporte a diferentes protocolos de rede.
Cacti	a. Não envia alerta, mas gera logs que contém as informações de erros. b. Desenha gráficos referentes ao uso de memória, tráfego na rede, espaço em disco etc. c. Possibilita criação de diferentes níveis de acesso aos dados coletados, por gestão de usuários. d. Possibilita expansão usando *plugins*, como o ***PHP Network***, que mostra um mapa da rede e o estado dos elementos.
Nagios	a. Monitora SMTP, POP3, HTTP, NNTP, ICMP, SNMP. b. Monitora recursos computacionais e equipamentos de rede. c. Realiza monitoração remota usando SSH ou SSL. d. Possibilita criação de scripts pelo próprio usuário. e. Faz resolução pró-ativas de problemas.

3 SISTEMA DE DIAGNÓSTICO INSTANTÂNEO - SDI

Neste capítulo será abordada a forma como o SDI funciona, suas principais funcionalidades uma análise sobre possibilidades de extensões. Em particular alguns aspectos importantes de sua implementação serão apresentados.

3.1 OBJETIVOS

De acordo com Ribas (2009), o Sistema de Diagnóstico Instantâneo é uma ferramenta útil no controle de informações computacionais de modo geral. Com ela, é possível fazer *(i)* um inventário completo dos *hardwares* de uma rede de computadores, seja local ou remota, *(ii)* é possível fazer um controle de processos dessas máquinas, *(iii)* identificar possíveis problemas e apresentar uma solução instantânea.

Também é possível integrar essa ferramenta com outros sistemas que apresentem a mesma proposta, que inclui o funcionamento baseado em *scripts* que serão distribuidamente executados pelo servidor nos clientes. Dessa forma, seria possível o administrador usar os recursos que esses sistemas pudessem oferecer de uma maneira útil para ele.

O SDI na verdade é uma plataforma para a execução dos programas e *scripts* desenvolvidos pelo administrador nas máquinas clientes (ou *hosts*), ou seja, o SDI executa os comandos para obtenção de informação dos dispositivos (*scripts*) de forma distribuída em todos os *hosts*, que retornam os dados solicitados. O SDI também oferece um *output* das informações obtidas, que são apresentadas em uma interface *Web* podendo ser disponibilizada na Internet.

> O Sistema de Diagnóstico Instantâneo (SDI) provê uma interface simples para observação dos dados coletados e meios rápidos e seguros para propagar automaticamente soluções desenvolvidas pelo administrador aos problemas encontrados em sua rede, isso integrado a um modelo de auto-gerenciamento e auto-configuração em grandes redes. (BONA et al, 2008, pg 1)

O SDI oferece uma grande liberdade ao administrador, de uma maneira tal, que ele tem a opção de desenvolver novas funcionalidades, ou funcionalidades específicas. Assim, o administrador não é limitado a usar somente as funcionalidades disponibilizadas pelos sistemas de monitoramento que existem no mercado.

3.2 FUNCIONALIDADES:

De acordo com Kurose (2006), em se tratando de redes de computadores, ou parques computacionais, um dos maiores desafios encontrado é o gerenciamento e suporte à (na maioria dos casos) enorme gama de computadores clientes, tendo em vista que por muitas vezes existe uma grande distância entre servidor e clientes, além da tarefa monótona de manter o inventário sempre atualizado e consistente.

> Quando centenas ou milhares de componentes são montados em conjunto por alguma organização para formar uma rede, não é surpreendente que ocasionalmente eles apresentem defeitos, que elementos da rede sejam mal configurados, que recursos da rede sejam utilizados excessivamente ou que componentes da rede simplesmente quebrem. (KUROSE, 2006, pg. 571)

Com o SDI, é fácil manter um inventário atualizado. As opções para execução periódica dos comandos é altamente configurável, podendo ser feita diariamente. Além disso, quando uma determinada situação crítica for atingida, o administrador é avisado imediatamente por *e-mail*. Essas e outras funcionalidades serão descritas a seguir.

3.2.1 Gerenciamento do Inventário de *Hardware*.

Na interface web do SDI é possível disponibilizar a capacidade de memória RAM, assim como velocidade, marca e modelo do processador, além dos dispositivos e *drivers* de rede, multimídia, vídeo e afins. De uma forma geral, isso é válido para todo dispositivo que for reconhecido pelo *kernel* do Linux.

De acordo com Negus (2006), no Linux, os dispositivos que são enxergados pelo núcleo do sistema, têm sua informação disponibilizada em arquivos textos que são atualizados continuamente. Muitos desses arquivos encontram-se em um diretório padrão (geralmente no "*/proc*"). Ou seja, essa informação está acessível para ser mostrada quando necessário. A figura 3-1 mostra um exemplo prático sobre a coleta de informações no *"/proc".*

```
1  # Print memory in MB
2  MEMORY="$(cat /proc/meminfo | grep MemTotal | awk '{print $2}')"
3  ((MEMORY=(MEMORY)/1024))
4
5  printf "RAMMEMORY+$MEMORY\n"
```

Figura 3-1: *Script* que imprime a quantidade total de memória de uma máquina em MB

Neste caso, foi escrito um *script*, em *Shell script*, pelo administrador, que busca a informação do total de memória existente no *host*. Esse filtro "*grep MemTotal"* é aplicado no arquivo "*/proc/meminfo*" que contém diferentes informações a respeito da memória física do sistema, como uso e capacidade por exemplo. Assim como o arquivo "*/proc/meminfo"* disponibiliza informações sobre memória, o arquivo "*/proc/cpuinfo"* também disponibiliza informações sobre CPU, além de muitos outros exemplos.

Além da opção dos arquivos usados pelo *kernel*, existem alguns *softwares* capazes de mostrar essas informações, como o *lspci* por exemplo. Segundo Negus (2006) *"/sbin/lspci"* é um comando disponibilizado pelo Linux que exibe informações gerais sobre o *hardware*, mais especificamente, esse comando cria uma lista de todos os dispositivos PCI existentes no computador.

Uma vez desenvolvido o *script*, o SDI se encarregará de abrir a conexão com a máquina cliente, executá-lo e mostrar a informação atualizada com uma periodicidade determinada pelo administrador.

3.2.2 Identificação e Correção de Falhas

Os *scripts* que forem criados, não necessariamente precisam ter como objetivo a coleta de dados (ou inventário) sobre o *hardware* do sistema computacional. Apesar deste não ser o objetivo principal do sistema, os *scripts* podem ser usados para efetuar controle de processos, controle de uso de memória, controle de uso de CPU, utilização de antivírus *etc*.

No SDI, para cada *script* ou programa criado, existe um arquivo *.po* (*parser object*) correspondente. Esse arquivo é responsável por determinar se o *script* executado terá seu resultado disponibilizado no site. No caso de ser um *script* de não-monitoramento, basta alterar o arquivo *.po* correspondente, inibindo a exposição *online* da informação, pois neste caso, não há informação alguma a ser mostrada, simplesmente deseja-se corrigir ou prevenir uma situação. Mais detalhes sobre o funcionamento do sistema serão dados na seção 3.3.

O conceito de falhas neste caso é um pouco mais flexível, e pode tomar um caráter subjetivo, fazendo com que o administrador seja o responsável por definir o que é ou deixa de ser uma falha. Tudo depende da necessidade do sistema monitorado como um todo.

Como um exemplo de *script* para identificação e correção de falhas, talvez fosse interessante, definir um programa que controlasse o número de processos correntes nos *hosts* (como mostrado na figura 3-2). Suponha que seja necessário em um sistema em rede que se tenha no máximo 100 processos rodando por máquina. Neste caso, é contado o número de processos (primeira linha) e comparado esse número com 100, se for maior, os últimos processos que foram iniciados serão terminados, e para cada processo finalizado, será enviado um *e-mail* avisando o ocorrido.

O SDI provê suporte para a realização de *scripts* tais qual o que está sendo demonstrado na figura 3-2, mas originalmente, os *scripts* usados pelo sistema, têm o objetivo da coleta de informações sobre a rede e/ou seus recursos computacionais.

```
#Max process running
NPROC=$(ps ax | wc -l')
LASTPROC=$(ps ax | tail -1 | awk '{print $1}')
while true
do
    if [ 100 -le $NPROC ]
    then
        echo $LASTPROC | mail -s "Process Killed" sdiufes@yahoo.com
        kill $LASTPROC
    else
        exit 0
    fi
done
```

Figura 3-2: *Script* que controla a quantidade de processos rodando em um sistema, máximo de 100 processos.

Alguns administradores podem achar que terminar o processo pode ser uma medida drástica. A solução neste caso seria fazer um *script* mais elaborado, que determinasse qual a classe de processos que poderia ser destruída em uma eventual necessidade, ou então, poderia adotar como solução padrão, o simples envio de *e-mail* para o administrador, passando para ele a responsabilidade do encerramento do processo. Como já foi dito anteriormente, tudo depende da necessidade da rede.

3.2.3 Categorização e Criação de Estados

Dentro do SDI, o conceito de *Classes* é definido para a organização das informações que chegam a cada momento no servidor. Segundo Ribas (2009), as classes são utilizadas para categorizar grupos de máquinas de acordo com determinadas características: Localização física, função, capacidade de processamento, ou qualquer outro tipo de identidade. Essa segregação permite uma personalização da administração além de facilitar a busca por máquinas.

Além do conceito de classes, existe o conceito de estados. Para explicar esse conceito, imagine como exemplo uma xícara de café, que pode estar: cheia, vazia, quente e fria. A todas essas características atribuídas à xícara, pode-se querer saber em qual estado ela encontra-se em um determinado momento.

Lembrando que alguns estados podem ser mutuamente exclusivos, como quente e frio. O mesmo esquema se aplica aos *hosts* do SDI.[6]

Um exemplo simples a ser citado, são os estados de *uptime* (tempo total no qual a máquina encontra-se ligada) alto e baixo que vem habilitado por padrão no SDI. Em muitos outros casos, o conceito de estados se aplica: Estado do RAID, Arquivos corrompidos, quantidade de processadores e memória *etc*. [6]

Segundo Ribas (2009), os sumários podem ser conceituados como um conjunto de estados. Sua função é manter as informações geradas pelos estados em uma página do SDI. A quantidade e nome dos sumários são personalizáveis, da mesma forma que sua associação com estados que representam.

Existem dentro da árvore de diretórios do SDI quatro pastas: "*summaries-available*", "*summaries-enabled*", "*states-available*", "*states-enabled*". As pastas com o sufixo *"available"* conterão os estados/sumários disponíveis, e as pastas com o sufixo *"enabled"* terão os estados/sumários habilitados. Cada vez que um Estado ou sumário é criado, é necessário ativá-los, como os *scripts*.

3.3 A ARQUITETURA DO SDI

O SDI foi desenvolvido com a intenção de facilitar sua extensão. Essa liberdade é representada por uma topologia simples, e por uma árvore de diretórios extensa, mas de fácil entendimento. Serão listados e explicados aqui os principais diretórios e arquivos de configuração necessários para compreender o funcionamento do sistema.

3.3.1 Topologia da Rede

A topologia que envolve a rede que possui o SDI como aplicativo de monitoramento, necessariamente é composta por um servidor e os *hosts*. A

[6] Exemplo dado por Vinicius Ruoso em <http://wiki.c3sl.ufpr.br/sdi/index.php/Criando_estados>, acessado em 08/2009

comunicação entre servidor e cliente é toda feita por meio de túneis SSH. O servidor SDI necessariamente tem que ser Linux, e será responsável *(i)* pela abertura dos túneis SSH em cada *host*, *(ii)* pela execução dos *script*s e também *(iii)* pela exposição das informações.

Segundo Harpers (2008), SSH (*Secure Shell*) é um sistema bastante seguro e escalável para autenticação de usuários e programas em outras máquinas ou redes de computadores. Além disso, ele fornece várias abordagens para redirecionamento, encaminhamento, restrição e negação de conexões.

Os clientes nesse sistema, assim como o servidor, têm que ser Linux, e são quase totalmente passivos. Apenas pequenas medidas de configuração devem ser tomadas neles. Uma dessas medidas é o cadastramento da chave pública gerada pelo "*ssh-keygen"* no servidor. Essa chave deve ser escrita no arquivo "*authorized_keys"* que se encontra no diretório *"~/.ssh"* do usuário SDI.

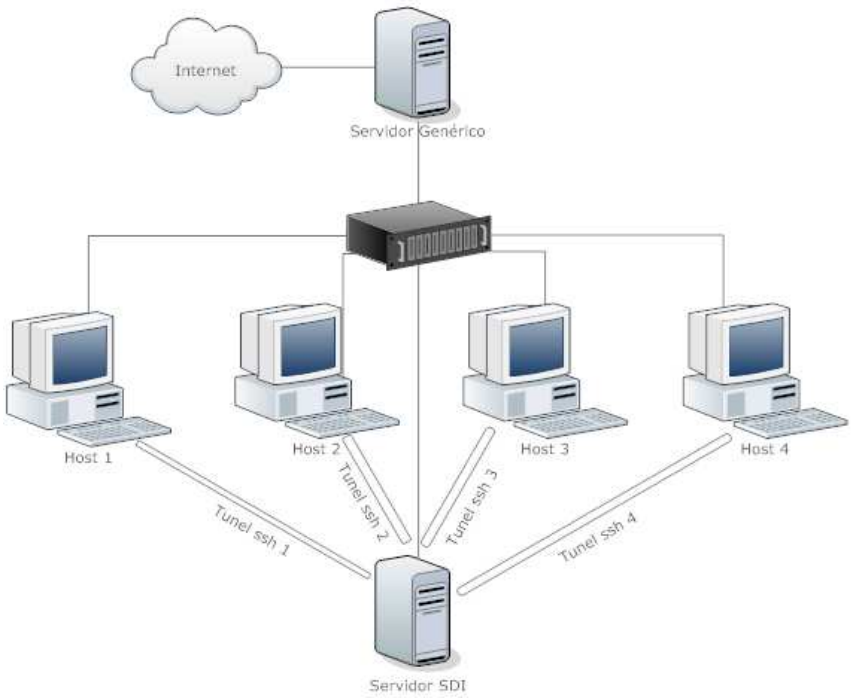

Figura 3-8: Topologia da rede no ambiente de testes do SDI usado no desenvolvimento deste trabalho.

O esquema montado na figura 3-8 foi baseado no ambiente de testes e desenvolvimento do SDI para este trabalho. É um esquema simples que segue o modelo de arquitetura cliente-servidor. Como todas as conexões entre

servidor e clientes é feita por meio de túneis SSH, se faz necessário cumprir o pré-requisito nos *hosts* de haver um servidor SSH, assim como um usuário para autenticação e um *shell* para execução dos *script*s.

3.3.2 A Árvore de Diretórios

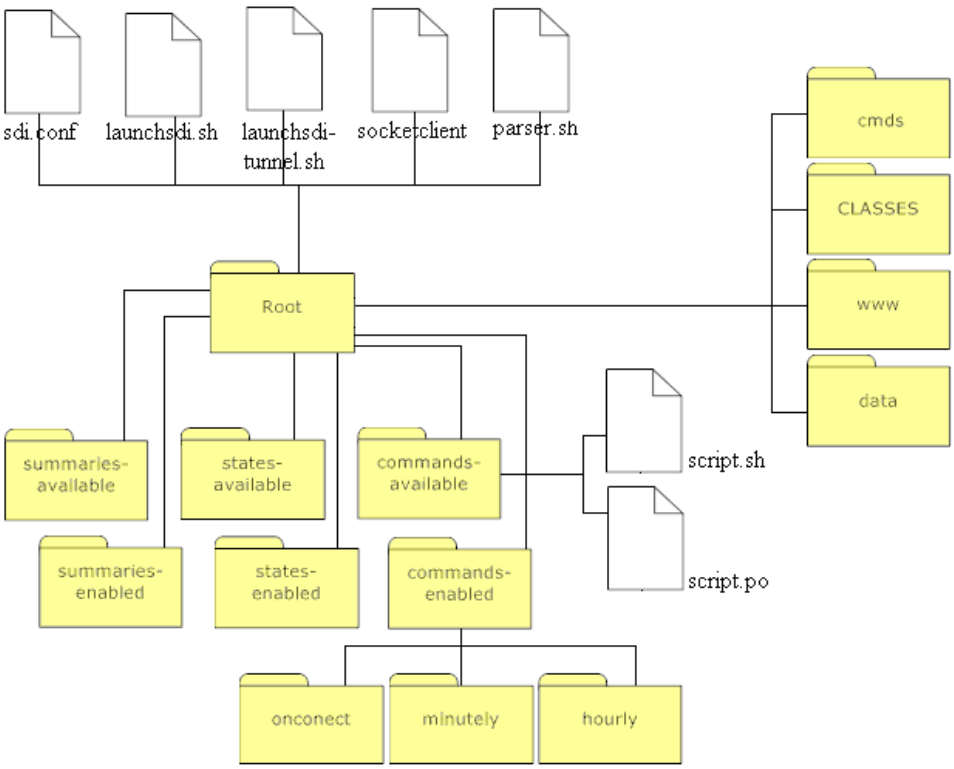

Figura 3-3: Árvore de diretório do SDI resumida

Na figura 3-3 é apresentada a árvore dos diretórios do SDI, que ilustra um esboço dos principais arquivos e diretórios que compõem a base do sistema. São arquivos e diretórios essenciais para o funcionamento ideal. Alguns dos diretórios e/ou arquivos que estão nessa árvore já foram citados antes, mas eles serão detalhados de forma a fazer entender o funcionamento geral do SDI

a) *States*-available

É a pasta que contém os arquivos de estado desenvolvidos pelo administrador. Esses arquivos terão como conteúdo a definição dos estados, que segue um padrão bem simples. Cada arquivo terá uma função

chamada ***getstateinfo()*** (representada na figura 3-4) que é única e fornece ao SDI as informações necessárias para gerenciamento dos estados.

As variáveis dessa função são: STITLE (título do estado); SDESCRIPTION (descrição do estado); SDEFCOLUMNS (colunas da tabela na interface *Web*) e SSUMARY (definição do significado do estado)**.**

```
1  function getstateinfo()
2  {
3      STITLE="Low uptime"
4      SDESCRIPTION="Hosts that have a low uptime"
5      SDEFCOLUMNS="Hostname,Uptime,Status"
6      SSUMARY="%d hosts with low uptime"
7  }
```

Figura 3-4: Função *getstateinfo()* presente nos arquivos de estado do SDI

b) *States-enabled*

Nesta pasta deverão ser criados *links* simbólicos para os estados existentes na pasta "*states-available*"*.* Ao realizar essa ação o administrador estaria "ativando" o estado.

c) *Summaries-availables/Summaries-enabled*

Funcionam de forma similar ao par anterior. O sumário é criado na pasta "*summaries-available*" e ativado na pasta "*summaries-enabled*". Os sumários deverão implementar a função ***getsummaryinfo()*** (representada na figura 3-5) que é única e fornece ao SDI as informações necessárias para criar os sumários. As variáveis dessa função são: SNAME (título do sumário) e STATES (vetor que contém os estados que compõe o sumário).

```
1  function getsummaryinfo()
2  {
3      SNAME="SUMMARY"
4      STATES[0]="lowuptime"
5      STATES[1]="highuptime"
6  }
```

Figura 3-5: Função *getsummaryinfo()* presente nos arquivos de sumário do SDI

d) *Commands*-available

Segue o mesmo padrão dos diretórios com o sufixo *"available"* anteriores. Todo e qualquer *script* que for criado para monitoramento dos *hosts* deve ser colocado nessa pasta. O SDI funciona muito bem com *script*s escritos em *bash*, mas nada impede que os *script*s sejam feitos em *python*, C, *perl* ou qualquer outra linguagem. Basta fazer com que o SDI os execute da forma certa nos *hosts*.

O que acontece é que como o SDI mantém um túnel SSH com o *host*, sempre aberto, um *Shell* é disponibilizado no *host* em questão, fazendo com que seja possível executar um binário ou até mesmo compilar um fonte e depois executar o binário gerado. Porem se faz necessário "copiar" os binários para os *hosts* e lá efetuar sua execução. Isso pode ser feito de duas formas. A primeira seria antes da execução dos *scripts*, copiar o binário ou fonte para o *host* (isso pode ser automatizado ou não) e posteriormente criar um *script* dentro da pasta atual que chame (execute/interprete) o binário ou fonte já existente no *host*. A segunda opção seria usar a interface "*sendfile*" presente no SDI.

Para um funcionamento correto as respostas das execuções dos *script*s devem seguir o padrão pré-definido do SDI. As respostas devem vir em apenas uma linha, contendo o nome do arquivo em letras maiúsculas, seguido do caractere "+" seguido da resposta. Ex: "NOME_DO_SCRIPT+resposta_esperada". Além dos *scripts*, nesta pasta também ficarão os arquivos .po correspondentes.

O *Parser Object* é responsável por determinar o comportamento da execução. Existem três funções no *parser object*, a primeira é ***updatedata()***, que atualiza o arquivo correspondente dentro do diretório "*data*", (mais detalhes sobre esses arquivos mais a frente). Essa função tem apenas uma variável: UPDATA (indica como o resultado do *script* será escrito no arquivo correspondente dentro do diretório "*data*").

A segunda função é ***www()*** responsável por atualizar a informação obtida na interface *Web* do SDI. Suas variáveis são: PVALUE (é o valor a ser inserido na coluna referente ao *script*), PSTATUS (cor do valor obtido, ex: *red, green, gray*), PSORTCUSTOM (recebe um número que irá definir sua posição na apresentação das colunas na interface *Web*), P*STATE*TYPE (controla os estados associados ao *script*).

A terceira e última função, é ***getcolumninfo()***. Essa função determina a representação da informação colhida, na interface *Web* do SDI. Ela apresenta duas variáveis: WEBINTERFACE (pode receber *true* ou *false*, dependo do desejo de mostrar ou não o resultado desse *script* no site), COLNAME (nome que a coluna irá receber na interface). A figura 3-6 mostra um exemplo de um *script* criado, seu *parser object* correspondente, e em seguida, sua amostragem na interface *Web* do SDI.

hd.sh

```
#!/bin/bash

BUSY=$(df -h | awk '{print $5}' | tail -1)
PARTIC=$(df -h | awk '{print $1}' | tail -1)

echo "HD+$PARTIC is $BUSY busy"
```

hd.po

```
hd_updatedata()
{
    UPDATA="$*"
}

hd_www()
{
    PVALUE="$*"
}

getcolumninfo()
{
    WEBINTERFACE=true
    COLNAME="HD busy"
}
```

Figura 3-6: *Script* simples e seu *parser object* correspondente.

No arquivo **hd.po** pode ser observado que existe o prefixo “hd_” nas funções. Na verdade esse prefixo deve receber o nome do script correspondente seguido de “_”. Esse esquema de funcionamento faz parte de uma convenção que auxilia o sistema na identificação dispositivo inventariado no momento da execução do comando.

O arquivo **hd.sh** é um *script* em *bash* que imprime a porcentagem de utilização da última partição mostrada no comando **df** (o comando **df** imprime as partições montadas no sistema). Neste caso o *parser object*

está usando apenas as variáveis obrigatórias nas funções. Observe agora na figura 3-7, os detalhes da resposta do *script* **hd.sh** e a forma como as informações mostradas foram definidas pelo *parser object*, como o nome da coluna por exemplo.

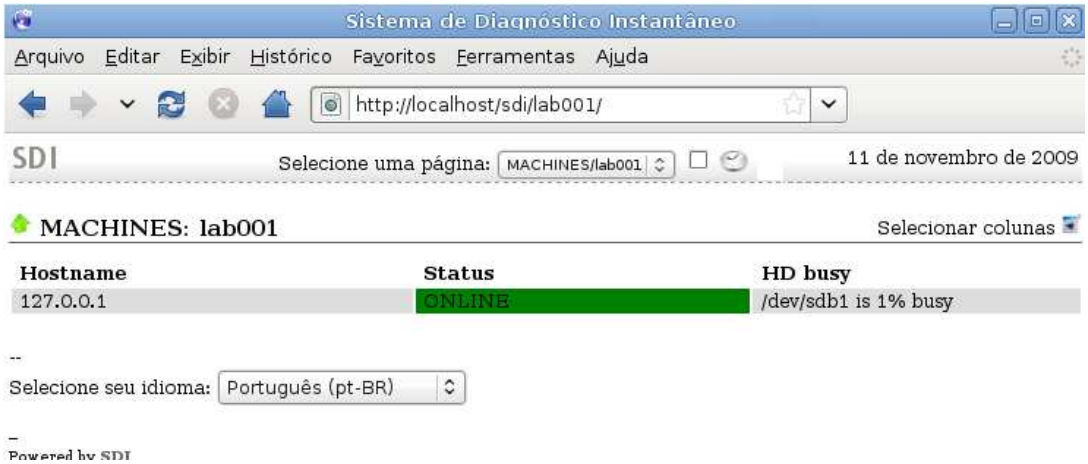

Figura 3-7: Interface *Web* do SDI exibindo o resultado do exemplo proposto

e) *Commands-enabled*

A informação como mostrada anteriormente, só estará disponível no site, se for criado um *link* simbólico para o *script* (que está na pasta "*commands-available*") neste diretório. A criação de um *link* simbólico no Linux é simples, basta usar o comando **ln -s.** Segundo Nemeth (1995) um *link* simbólico aponta para um arquivo pelo nome. Quando o *kernel* precisa abrir ou passar por um *link* simbólico e viabiliza o caminho armazenado como o conteúdo do *link*.

Dentro dessa pasta, existem os seguintes diretórios: "***onconnect*", "*minutely*", "*hourly*", "*daily*"** e "***montly***". Os *links* que estiverem nesses diretórios serão executados com a freqüência exata que o nome sugere. Isso acontece, pois o SDI toma as medidas cabíveis para acrescentar a periodicidade de execução do *script* na *cron tab*.

De acordo com Nemeth (2009) no UNIX, execuções periódicas são manipuladas pelo *cron*. O *cron* lê um ou mais arquivos de configuração contendo listas de linhas de comando, hora a qual elas serão invocadas e usuários com a permissão para essa execução.

f) CLASSES

Esse diretório irá conter o cadastro dos *hosts* ligados ao SDI. É nele que o sistema irá buscar os nomes ou endereços das máquinas clientes. Aqui, o conceito de classificação de clientes do SDI será definido. Suponha que o SDI esteja monitorando as máquinas de mais de um laboratório. Para cada laboratório existirá um arquivo texto contendo os nomes/IPs dos *hosts* separados por quebra de linha.

A categorização não precisa ser feita necessariamente pela localização, ela poderia ser feita por serviços que um determinado conjunto de máquinas desempenha, ou por *hardware*, ou por qualquer outro item de classificação, tudo depende da necessidade da rede.

g) Data

Os resultados dos *scripts* precisam estar disponíveis em algum lugar após a coleta das informações, principalmente para o módulo do sistema responsável pela exposição dos dados no site. Este local é a pasta “*data*”.

Para cada *host* existirá uma pasta dentro desse diretório, com os *logs* e informações obtidas por cada *script* separadamente em arquivos textos. Com isso, será mantido um controle histórico/atual da informação.

h) www

Esse diretório, contem os arquivos do site. Ele é gerado automaticamente pelo módulo *sdiWeb* após o lançamento do SDI. O diretório **www** não precisaria existir, pois no arquivo **sdi.conf** seria possível modificar o local onde seriam escritos os arquivos que montariam o site, no caso, o local citado poderia ser a pasta padrão do *apache*, ou qualquer outro servidor *Web* em uso. É claro que isso só seria possível se o usuário do SDI tivesse permissão de escrita na pasta (“/var/www” para sistemas Debian).

i) Cmds

Nesse diretório estão armazenados os arquivos de comando, que estão disponíveis para serem rodados individualmente em um determinado *host***,** ou para todos de uma vez. No caso do *script* rodando individualmente, o resposta vai para um arquivo de *log*.

j) Sdi.conf

É o arquivo de configuração do SDI. Nele são determinadas algumas ações que o sistema deve tomar na hora em que for lançado. Nele também são definidos quem serão os diretórios "*data*", "*www*" e "*CLASSES*". Ele também é responsável por definir os módulos de envio e recebimento de mensagem.

Caso o administrador não tenha interesse na interface *Web*, ou seja, quer usar o sdi para outro fim que não seja o monitoramento, ele pode definir isso no sdi.conf, que é quem vai dizer se o site vai estar habilitado ou não, além de prover configuração para algumas colunas padrões na interface.

k) *Parser*.sh

É o arquivo responsável por entender os arquivos *parser object*. Ele toma posse da mensagem que o *host* retornou e decide o que fazer com ela. Se existe um arquivo .po associado àquela mensagem, ele formata, cria as páginas *Web*, gera os estados *etc*. Caso nenhum .po exista, o *parser* simplesmente imprime no *log* genérico de mensagens daquele *host*.

l) *Socketclient*.c

Quando é enviado um comando para todos os *hosts* ao mesmo tempo, em tese a resposta volta em mais ou menos ao mesmo tempo de todos os *hosts*. Caso o número de *hosts* seja muito alto, existiriam muitos *parser* aplicando os filtros e executando o .po para cada uma das mensagens. Isso exigiria muito do processador que provavelmente não teria um bom *throuput* das mensagens.

A intenção do *socketclient* é evitar esse problema, pois ele limita a quantidade de *parser* executando ao mesmo tempo, neste caso, antes do

parser iniciar o processo com a mensagem, ele aguarda autorização do *socketclient* que se conecta a uma fila. Enquanto a fila de *parsers* executando for maior que um limiar (*default* 100) o *socketclient* fica bloqueado até alcançar a cabeça da fila.

m) *Launchsdi.sh*

Esse *script* feito em *bash* é responsável por lançar o SDI. Durante sua execução, ele chama o *script* que irá criar os estados, inicia o *socketclient*, e chama o *script launchsditunnel.sh*.

n) *Launchsditunnel.sh*

Uma das funções desse *script* é abrir os túneis SSH, que uma vez abertos, serão mantidos assim até o encerramento do sistema. Encerramento esse, também feito por esse *script*.

Existem algumas opções de execução desse *script*, são elas: *--kill=HOST* (finaliza o tunnel SDI para o *host*); *--killall* (fecha todos os túneis e pára a execução do SDI); *--reload-po* (força um recarregamento dos *parsers objects*); *--reload-states* (força um recarregamento dos arquivos de estados).

Nesse arquivo existe a função SDITUNNEL, que executa um ***tail -f*** nos arquivos de comandos (cmds/$*HOST* para um *host* específico, ou cmds/general para enviar para todos). Esses comandos são enviados via *pipe* para o SSH, que por sua vez executa um "*bash* -s" no lado do *host*.

O ***tail -f*** mantém o túnel SSH aberto aguardando por novos comandos, e um *loop* infinito no início da função SDITUNNEL garante que o SSH volte a ser aberto em até 15 minutos, caso caia por algum motivo.

3.3.3 Funcionamento dos Componentes do SDI

Agora que a árvore de diretórios do SDI é conhecida e a topologia interna está mais evidenciada, pode-se descrever de forma resumida o funcionamento do sistema para execução dos *script*s criados nos servidor.

A primeira medida é criar o *script* e seu *parser object* correspondente. Feito isso, deve-se habilitar o *script* criando um *link* simbólico em um dos diretórios de periodicidade que se encontram na pasta "*commands-enabled*".

Quando o SDI for lançado, o *launchsdi* efetuará uma busca pelos IPs/nomes dos *hosts* no diretório "*CLASSES*", e posteriormente irá chamar o *launchsditunnel* passando os *hosts* como parâmetro. Então o *launchsditunnel* se encarregará de abrir os túneis SSH e executar o *socketclient*.

Uma vez os túneis abertos, o *launchsditunnel* inicia a execução dos *script*s nos *hosts* e escreve os resultados obtidos no diretório "*data*". Agora o *parser* será iniciado, e começará uma execução nos arquivos .po. O resultado dessa execução é a montagem dos arquivos do site, que irão para a pasta "*www*".

A pasta "*www*" é o objetivo final do SDI. Ela irá conter todas as informações coletadas pelos *script*s da forma como o *parser object* foi configurado para elas serem mostradas. Mas o resultado final também inclui a geração dos *logs* e das informações escritas no diretório "*data*".

3.4 CONSIDERAÇÕES SOBRE O CAPÍTULO

O SDI apresenta um código fonte bastante distribuído e dividido em diferentes linguagens de programação (C, bash e python) o que dificulta de certa forma, seu entendimento, mas isso é compensado, pois de acordo com Ribas (2009), o sistema se mostrou efetivo e de grande valia para o projeto para o qual ele foi desenvolvido. Além disso, o SDI transformou a gerência da rede num processo mais simples, menos trabalhoso e mais ágil na resolução de problemas.

Segundo Ribas (2009), o diferencial do SDI quanto às aplicações usadas é o processo de organização das informações recebidas, separada em histórico e atual. A disposição de uma API flexível para criação de novos *scripts*, e a visualização dos dados de modo personalizável e em tempo real, são as características em que o sistema é baseado.

O mais importante, é a grande liberdade proporcionada. Os *script*s são elaborados de acordo com a necessidade presente. Muitas vezes um administrador ao usar um sistema diz que determinada informação mostrada poderia ser dada de um jeito diferente. Com o SDI ele é quem defini isso.

4 IMPLANTAÇÃO E EXTENSÃO DO SDI

O objetivo inicial deste trabalho foi fazer um estudo aprofundado sobre o SDI realizando em seguida uma avaliação sobre o mesmo ao implantá-lo no Laboratório de Pesquisas em Redes e Multimídia (LPRM) da UFES. Com esses estudos e com os primeiros testes realizados verificou-se que as características que o SDI apresenta poderiam ser bem aproveitadas no laboratório. Mas para um melhor aproveitamento do sistema algumas extensões foram realizadas a fim de abranger os tipos de sistemas suportados, elevar o nível de resolução automática de problemas e integração das funcionalidades de outras ferramentas de gerência.

4.1 REQUISITOS DO LPRM

O LPRM foi o ambiente utilizado para a implantação, avaliação e desenvolvimento sobre o SDI. O LPRM é um laboratório de desenvolvimento de projetos de pesquisa do Departamento de Informática da UFES, sendo um dos laboratórios ligados ao Programa de Pós-Graduação em Informática da UFES. Durante a utilização do SDI no LPRM verificou-se que sua utilização poderia ser melhorada com o atendimento de alguns requisitos.

4.1.1 Portabilidade.

Para funcionamento do SDI é necessário que se cumpram alguns pré-requisitos nos *hosts*: Ter um servidor SSH, a existência de um *shell* para a execução dos *script*s e um usuário para a autenticação na conexão SSH. O SDI é uma ferramenta do projeto Paraná Digital, usado para monitorar todos os 2.100 servidores da rede de escolas do governo do Paraná.

> O Paraná Digital – PRD – é uma inovadora experiência no âmbito da inclusão digital. Por meio de uma parceria entre Universidade Federal do Paraná (UFPR), governo estadual e outras instituições, estão implementando laboratórios de informática em todas as escolas da rede pública do Paraná. (SILVEIRA, 2009, pg. 5)

Segundo Silveira (2009) cada um dos servidores das escolas envolvidas no PRD usa Linux como SO. Tendo isso em vista os desenvolvedores do SDI implementaram o sistema para funcionar nativamente em sistemas Linux.

No LPRM muitos computadores têm Windows instalado como sistema operacional principal, e essa foi a maior barreira encontrada para a implantação do SDI, visto que o sistema operacional citado não cumpre nenhum dos quesitos obrigatórios para o correto funcionamento do sistema.

4.1.2 Utilização de Outras Ferramentas de Gerência.

No servidor principal do LPRM já existiam duas ferramentas de gerenciamento particularmente aplicadas ao monitoramento da rede: Cacti e Nagios. Concluiu-se que seria importante estender o SDI permitindo uma integração com essas ferramentas. Desta forma, novas funcionalidades seriam incorporadas ao sistema, além de permitir um aproveitamento maior do ferramental já instalado no laboratório, oferecendo um acesso integrado ao mesmo.

4.1.3 Detecção de Falhas Iminentes

O ambiente computacional do LPRM é bem propício ao acontecimento de problemas, pois os computadores são utilizados para aplicações diferentes por usuários diferentes a todo tempo.

Em alguns casos, os computadores permanecem ligados por semanas, executando alguma aplicação, para ser acessado remotamente ou por comodidade. Tendo em vista essa situação, alguns aspectos poderiam ser observados com o passar do tempo para serem evitados se ocorressem com uma repetição constante. Por exemplo, um dos aspectos observados em uma dessas máquinas que permaneciam ligadas por um longo período, foi uma elevação na temperatura do processador, e o superaquecimento, segundo Morimoto (2002), perdurando por muito tempo, pode danificar o componente.

Além do problema do processador uma série de problemas iminentes que trariam prejuízo computacional, como destruição de hardware ou problemas de desempenho, que engloba: exaustivo uso de *swap*, processos tomando posse de 100% da CPU durante muito tempo, discos altamente fragmentados ou com blocos corrompidos, dentre outros.

Portando, torna-se importante não somente resolver os problemas considerados críticos, mas também criar estatísticas da ocorrência desses problemas. Essas estatísticas podem auxiliar em um estudo mais profundo sobre os principais fatores responsáveis pela ocorrência dos problemas.

4.2 EXTENSÕES REALIZADAS

Algumas extensões foram implementadas no SDI de forma a atender os requisitos apresentados na seção anterior. Desta forma pôde-se adaptar o SDI às necessidades do LPRM, conforme detalhado a seguir.

4.2.1 Portabilidade

Para o SDI funcionar tendo como *host* uma máquina Windows, foi necessário transpor as barreiras dos pré-requisitos impostos pelo sistema. Para fazer isso, algumas propostas surgiram, e foram analisadas.

A primeira tentativa foi implementar nos clientes um servidor de mensagens instantâneas que usasse o protocolo *jabber* de comunicação. O que mudaria drasticamente a forma de atuação do SDI.

Essa idéia foi concebida em conjunto com os desenvolvedores do SDI como alternativa de comunicação entre sistemas Windows e Linux, tendo em vista que o SSH não é um aplicativo nativo em sistemas Windows. Neste caso, o *jabber* deveria substituir o SSH.

De acordo com Shigeoka (2002), o protocolo implementado pelo *jabber* representa uma solução simples, mas poderosa para o envio e recebimento de mensagens. Neste caso, o servidor tem a função de prestar serviços aos clientes *jabber*, enquanto os clientes normalmente atuam como agente de exibição de informações para os usuários, além de responder às entradas deles. Nessa configuração, o "usuário" seria o SDI, isto é o servidor SDI executaria o cliente *jabber*.

O problema em usar essa opção, é que os papéis de cliente e servidor SDI, se inverteriam. Agora os *host*s implementariam e executariam os *scripts*, e a periodicidade dessa execução também seria determinada por eles. Posteriormente, os *hosts* deveriam enviar as informações para o servidor. Dessa forma a execução do SDI seria completamente descentralizada, e a filosofia de "quanto menos esforço houver no cliente, melhor" deixaria de existir. Tendo em vista essa grande mudança de abordagem, o modelo de troca de mensagens utilizando o protocolo *jabber* foi abandonado.

A segunda tentativa de portar o SDI envolveu buscar um servidor SSH para Windows. Pensou-se então na solução oferecida pelo *Openssh for Windows*.

> O "*OpenSSH for Windows*" contém o servidor OpenSSH, provindo do pacote *Cygwin*, necessário para executar o servidor SSH. *Cygwin* é um ambiente Linux-like para Windows ... copSSH é fácil de instalar, e os pacotes diferentes têm licenças livres. Plataformas suportadas são Windows NT/2000/XP/2003/Vista. (HARPERS et al, 2008, pg. 22)

Segundo Barret e Silverman (2001), com esse *software* é possível abrir conexões SSH entre sistemas Linux e Windows sem maiores problemas, além disso, também é possível usar trocas de chaves públicas para autenticação sem senha.[7]

A interface de interação disponibilizada depois da abertura dos túneis SSH é o *prompt do DOS*, e essa foi a maior razão pela qual essa tentativa foi abandonada. Além, da limitação do *prompt do DOS* em relação ao *shell* do

[7] O mecanismo de SSH atuante no SDI exige o funcionamento baseado em trocas de chave, evitando a digitação de senha (para cada *host* monitorado) no momento da abertura do túnel.

Linux, tem que ser levado em consideração que a maioria dos *scripts* usados pelo SDI, são escritos em *bash*, ou seja, precisam de uma plataforma *shell* para serem executados. Nessa situação, tem-se apenas o usuário para autenticação e uma conexão SSH, mas não há suporte a uma plataforma para execução dos *script*s (i.e. o *Shell*).

A terceira possibilidade foi a busca de um *shell* para Windows, chegando-se então ao *software Cygwin*.

> As ferramentas do *Cygwin* são as portas das mais populares ferramentas desenvolvidas pelo GNU e utilitários para Windows. Elas funcionam através do uso da biblioteca *Cygwin* que fornece as chamadas do sistema UNIX e o ambiente que esses programas requerem. (VINSCHEN et al, 2001, pg 1)

De acordo com Vinschen (2001), o *Cygwin* apresenta uma DLL que age como uma API de emulação do Linux, provendo todas as funcionalidades do *shell* que o SDI necessita para funcionar. Caso alguma ferramenta necessária para a execução de determinado *script* não exista no *Cygwin*, ela pode ser facilmente disponibilizada em uma posterior instalação/atualização do *Cygwin*.

Outra vantagem de usar o *Cygwin*, é que ele emula uma árvore de diretórios idêntica à do Linux no Windows. Com isso, não existiriam maiores problemas na execução de *script*s que envolvam diretórios específicos e arquivos ligados ao *kernel* do sistema.

Adotando-se a solução baseada no Cygwin os pré-requisitos que os *hosts* devem cumprir, passam a ser diferentes para Windows e Linux. No caso do Windows, deve-se haver um usuário para autenticação no SSH e o *Cygwin*.[8]

Além dos pacotes que já são instalados com a versão padrão do *Cygwin*, torna-se necessário adicionar o pacote *openssh-server*, que funcionará como o servidor SSH em cada *host* Windows. Depois da instalação no *host*, é necessário realizar algumas configurações que serão detalhadas a seguir.

[8] Está disponibilizada no site <www.lprm.inf.ufes.br/*Cygwin*_lprm> uma versão do *Cygwin* já com todos os pacotes necessários para o funcionamento do SDI.

a) Ambiente do Windows

Somente por precaução deve-se acrescentar o caminho “C:*Cygwin*\bin”no *path* para que não haja maiores problemas com a execução dos binários do *Cygwin*. Mesmo no Windows, também é necessária a criação de um usuário para o SDI, que pode ser local, ou um usuário pertencente ao domínio o qual se encontra o *host*.

b) Configuração do Openssh-Server

Da mesma forma que existe um "/*etc*/passwd" e um “/*etc*/group” no Linux, no *Cygwin* existirá algo similar. Esses arquivos devem ser modificados para permitir a autenticação do usuário na criação do túnel SSH. Para a realização dessa tarefa, os comandos “*mkpasswd*” e “*mkgroup*” terão que ser executados.

Depois de criados os arquivos *passwd* e *group*, eles deverão possuir permissão de leitura, pois o próximo passo da configuração é a execução do comando “*ssh-host-config*” que irá usar esses arquivos na preparação do *host* para o funcionamento do SSH. O *ssh-host-config* se encarregará de criar arquivos de configurações e diretórios necessários e conceder privilégios aos usuários.

Finalmente, pode-se dar inicio o servidor SSH com o comando “*cygrunsrv -S sshd*”. *Cygrunsrv* é o comando que executa os serviços presentes no *Cygwin*, e *sshd* é o nome do serviço a ser executado. No caso este serviço é o *daemon* do Openssh-Server.

Os comandos que iniciam os serviços, como é o caso do *cygrunsrv* precisam ser executados apenas uma vez, mesmo que o sistema seja desligado ou reiniciado, pois uma vez iniciado o serviço, o servidor SSH será levantado automaticamente com o inicio do sistema.

c) Firewall do Windows

O SSH usa a porta 22 como padrão para abertura de conexão. Mas isso pode ser alterado no arquivo "*/etc/sshd_config*". Seja qual for a porta utilizada, ela deve ser liberada no *firewall* do Windows, que certamente tentará bloquear o acesso.

Transpondo todas essas configurações, pode-se acrescentar o novo *host* Windows, adicionando o novo IP/nome em uma das classes do SDI, como se ele fosse um Linux. Agora o esquema de funcionamento do SDI terá uma camada a mais na execução para os *hosts* Windows. A figura 4-1 mostra uma estrutura de rede adequada à portabilidade criada, tendo os *hosts* 1 e 4 o Windows como seu sistema operacional.

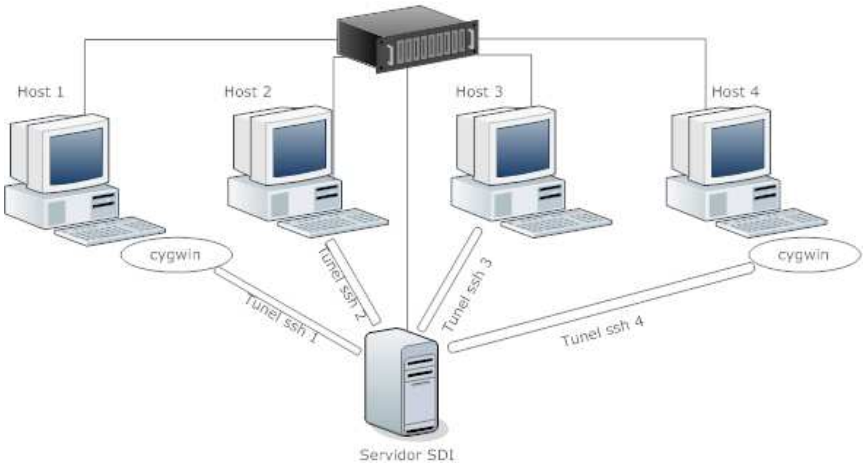

Figura 4-1: Topologia da rede incluindo o *Cygwin* como mediador das conexões SSH com Windows

4.2.2 Integração do SDI com a Ferramenta *Nagios.*

A idéia da realização da integração é fazer com que o SDI possa trabalhar em conjunto com qualquer sistema que tenha um funcionamento semelhante ao dele. A ferramenta *Nagios* foi aproveitada por isso e pelo fato de ela já estar instala e configurada no servidor principal do LPRM. Além disso, ela também usa *script*s para colher as informações, e as respostas também são dadas em apenas uma linha. Porém ainda é possível que o SDI seja integrado com qualquer outra ferramenta que siga a proposta apresentada pelo SDI/Nagios.

> O Nagios é uma aplicação desenvolvida para monitoramento de rede. Ele pode monitorar *hosts* e serviços, enviando notificações de eventos. Ele foi originalmente desenvolvido para Linux, embora também funcione com Windows. (KOCH, 2008, pg 25)

De acordo com Koch (2008), assim como o *Nagios* o *Cacti* é uma ferramenta de monitoramento, mas seu enfoque principal é o gerenciamento de contabilização. Neste caso os *scripts* que o *Cacti* poderia oferecer ao SDI o *Nagios* já oferecia, além disso, o esquema de funcionamento do Cacti não se adequaria tão bem como o do Nagios.

> Escrito em PHP, o Cacti atua em conjunto com o RRDtool, um sistema capaz de reproduzir em gráficos as informações dos elementos monitorados. Exemplos de tipo de informações que podem ser monitoradas são: largura de banda, requisições HTTP, média de processos, utilização de memória, entre outras. (KOCH, 2008, pg 27)

Para realizar a Integração entre o SDI e o Nagios, foi necessária a criação de um módulo à parte (*integration*) que é chamado no lançamento do SDI. Esse novo módulo deve ser ativado no arquivo "*sdi.conf*", onde também deve-se colocar o endereço do servidor Nagios.

Ao ser executado, o *launchsdi.sh* verificará junto ao "*sdi.conf*" se o modo de integração está habilitado. Se estiver, ele executará o módulo de integração do *Nagios* que se encontra dentro do diretório "*integration*".

Dentro do diretório "*integration"* existe uma pasta *Nagios* com os arquivos necessários para execução da integração. No conteúdo dessa pasta existe um arquivo de execução (*.sh*) um arquivo de configuração (*.conf*) e um diretório com os binários obtidos no servidor da ferramenta integrada (*remote_scripts*).

O arquivo de execução será responsável principalmente pela disponibilização, cópia e distribuição de *scripts* provenientes da ferramenta integrada. No arquivo de configuração deverá ser nomeado o diretório onde se encontram os *scripts* da ferramenta integrada, o diretório para onde esses *scripts* serão copiados, e todos os outros diretórios que serão úteis na realização da integração.

No caso do Nagios, o arquivo “*Nagios.conf*” conterá a definição da localização dos diretórios que serão usados na atuação desse módulo, e também a definição do endereço do servidor *Nagios*. Essas informações serão usadas pelo arquivo “*Nagios.sh*”.

O “*Nagios.sh*” é o *script* responsável pelo funcionamento ideal da integração com o *Nagios*. A primeira medida tomada por esse arquivo é identificar quem é o servidor *Nagios* e listar os binários que ele pode oferecer para o SDI. Posteriormente ele aguarda o *input* do usuário para escolher os binários que o servidor *Nagios* está oferecendo.

Depois que os binários oferecidos forem escolhidos, o “*Nagios.sh*” irá automaticamente copiá-los para a pasta “*remote_scripts*”, criar um *script*, seu *parser object* correspondente na pasta “*commands-available*” e o *link* simbólico na pasta “*commands-enabled*” para ativação o *script* criado.

A criação automática dos *script*s citados anteriormente é feita baseada em um padrão pré-definido. Quando o *script* criado (na pasta “*commands-available*”) for chamado, ele irá executar o binário provindo do servidor *Nagios* e imprimirá a resposta no formato exigido pelo SDI. Para o caso do *parser object*, foi disponibilizado um arquivo “*default.po*”. Esse arquivo apresenta somente as opções padrões habilitadas (como a opção do nome da coluna, por exemplo).

O *default.po* será copiado e modificado de acordo com o *script* relacionado. As modificações incluem alterar os nomes das funções (“*default*_updatedata()” para “*nome-do-script*_updatedata()”, por exemplo) e mudar o nome da cópia do arquivo (“*default.po”* para “*nome_do_script*.po”). Para a criação do *link* simbólico não são necessários grandes esforços, basta executar o comando “*ln –s*” na pasta “*commands-enabled*”.

Finalmente, o “*Nagios.sh*” copiará os binários (que agora estão no diretório “*remote_scripts*”) para uma pasta nos *hosts*. Essa pasta deverá ser pré-definida no arquivo *Nagios.conf*. Os binários copiados serão chamados na execução do *script* que foi gerado automaticamente pelo “*Nagios.sh*”.

Para um melhor entendimento, será exemplificada uma situação na qual existe um binário chamado “*check_tcp*”, disponível no servidor Nagios, que um administrador desejasse que fosse usado pelo SDI.

O SDI será lançado, tendo o modo de integração ativado. Durante o lançamento será perguntado qual binário do *Nagios* deseja-se executar no SDI. Apenas a opção do “*check_tcp*” será escolhida. Feita a escolha, o “*check_tcp*” será copiado para a pasta “*remote_scripts*”, e o módulo de integração criará dois arquivos na pasta “*commands-available*”: “*check_tcp.sh*” e “*check_tcp.po*”. Também será criado o *link* simbólico na pasta “*commands-enabled*”.

O “*check_tcp.sh*” foi criado somente para executar o comando “/tmp/sdi/*integration*/*check_tcp*” e o “*check_tcp.po*” está configurado de modo padrão, para a exibição dos dados. Agora o arquivo “*remote_scripts/check_tcp*”, será copiado para a pasta “*/tmp/sdi/integration*” no *host*. Essa pasta foi pré-definida no arquivo “*Nagios.conf*”.

O lançamento do SDI continuará normalmente, e quando chegar o momento de executar os *scripts* nos *host*s, a configuração para execução do “*check_tcp*” já terá sido preparada pelo módulo de integração.

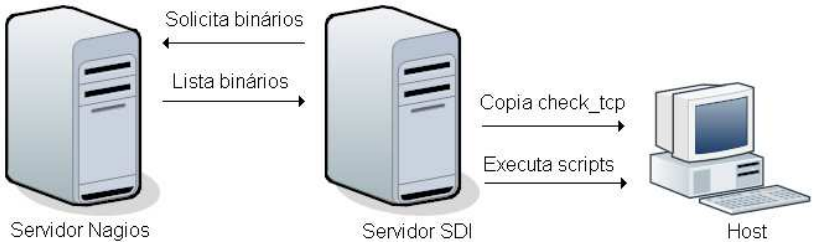

Figura 4-2: *Timelife* da execução do exemplo da utilização do *check_tcp.*

O *Nagios* e o SDI não precisam necessariamente estar em servidores diferentes, como mostra a figura 4-2. Eles podem ser executados em um mesmo computador, basta definir o endereço do servidor *Nagios* como “*localhost*” no arquivo “sdi.conf”. Além disso, também será necessário definir corretamente a pasta dos *script*s do *Nagios* no arquivo “Nagios.conf” localizado na pasta “*integration*”.

4.2.3 Detecção/Correção de Falhas

Novos *scripts* foram desenvolvidos com a finalidade de melhorar o funcionamento das máquinas do LPRM. Esses *scripts* têm a intenção de analisar o *host* e definir se existe alguma situação crítica. Na existência desta situação, o *script* deve tomar alguma providência.

Dentro do diretório raiz do SDI foi criada uma pasta "*failure-detection*" onde os *scripts* de correção de erros deverão ficar. Para o funcionamento desses *scripts*, é necessário que eles possuam um arquivo .po correspondente. Neste caso, as informações tratadas não precisam de um retorno, visto que a tarefa realizada será de manutenção e não de monitoramento. Também é necessário que sejam ativados na pasta "*Commands-enabled*", assim como os *scripts* de monitoramento.

No caso do ambiente do LPRM, a idéia foi criação de *scripts* que analisassem uso de memória RAM e uso de CPU. Isso porque as máquinas não apresentam um *hardware* com altas capacidades, e o uso indiscriminado dos recursos da máquina poderiam a deixar muito lenta.

Como agora *hosts* Windows também estão sendo gerenciados, é fato que algumas tarefas de manutenção poderiam ser automatizadas. Algumas propostas que poderiam ser adotadas, seria o uso do *scandisk* para verificação de blocos defeituosos e a desfragmentação nos discos, e até mesmo a realização de varreduras periódicas de antivírus.

Uma das necessidades dos computadores do LPRM é manter os discos dos computadores sempre desfragmentados. Para essa situação foi desenvolvido o *script* "*defrag.sh*" (Anexo XX) executado semanalmente, que verifica a integridade do HD, corrige os erros encontrados na verificação e o desfragmenta caso o disco esteja mais de 10% fragmentado.

Foi criada uma classe de processos chamados de “intocáveis”. São processos primordiais para o funcionamento de alguns serviços, como o apache por exemplo.

A proposta de melhoria do uso da CPU vem com o *script* “*processor.sh*” (Anexo XX). Esse *script* tem a função de controlar o uso da CPU. Se um processo não “intocável” tomar mais de 90% da CPU, durante um período maior que dez minutos, ele será terminado, e um *e-mail* será automaticamente enviado para o administrador notificando o acontecimento. Vale ressaltar que os valores de tempo e uso da CPU poderiam ser facilmente alterados, e a proposta de finalização do processo poderia ser mudada. Tudo depende da necessidade da rede.

Para o caso da utilização de memória, foi desenvolvido um *script* com a mesma finalidade do *script* desenvolvido para o uso da CPU. O arquivo “*memory.sh*” tem a função de encerrar o processo não “intocável” que estiver usando mais de 60% da capacidade de memória da máquina. Neste caso não importa o tempo em que o processo esteja ativo, tendo em vista que o uso abusivo de memória deixa o sistema consideravelmente lento.

4.3 CONSIDERAÇÕES SOBRE O CAPÍTULO

Durante a execução dos trabalhos para a extensão do SDI, muitas dificuldades foram encontradas, principalmente com relação à aplicação da portabilidade no sistema. Em particular, observou-se que os tempos contados com a execução dos serviços eram muito distintos mesmo sendo executados da mesma forma para ambos os sistemas operacionais envolvidos.

Por exemplo, o tempo de abertura de conexão envolvendo Windows era muito maior do que o tempo envolvendo Linux, mas a execução de determinados *scripts* era mais rápida quando feita em Windows do que em Linux, sendo que os testes foram realizados em um mesmo *hardware* usando *dual boot*.

No princípio, enquanto os tempos eram analisados, determinadas coletas eram julgadas falhas e posteriormente descartadas, mas na verdade elas estavam corretas, mas ainda não poderiam ser entendidas devido às descobertas que aconteceriam posteriormente.

Também existia o problema de haver pouca documentação referente ao SDI, por este motivo as dificuldades se agravaram ainda mais. A grande fonte de informação para a execução deste trabalho veio da parte dos próprios desenvolvedores, que se disponibilizaram a esclarecer muitas questões que estavam pendentes.

Como um dos resultados desse desenvolvimento, foi definido um artigo no Wiki do SDI documentando como pode ser efetuada a portabilidade no SDI. Trata-se de um *howto*[9] detalhado, com uma demonstração da realização da configuração.

[9] O artigo encontra-se no site <http://wiki.c3sl.ufpr.br/sdi/index.php/ Portando_o_SDI>.

5 AVALIAÇÃO DA PROPOSTA DE EXTENSÃO

Uma vez implementadas as extensões descritas no capítulo precedente, torna-se necessária uma avaliação do que foi desenvolvido para determinar se as essas extensões podem comprometer a abordagem proposta pelo SDI.

A proposta de executar *script*s de correção de falhas, não muda o funcionamento normal do SDI, apenas mantém a utilização dos recursos que o sistema oferece, para executar uma nova categoria de *scripts*. Portanto, não foi necessária a realização da avaliação para este caso.

Da mesma forma, a proposta da integração do SDI com o *Nagios* não altera o funcionamento do sistema. Com essa proposta, apenas é incluído um novo passo durante o lançamento do SDI, que praticamente se resume à execução do *script Nagios.sh* no diretório "*integration*".

5.1 OBJETIVO

Tendo em vista que uma nova camada (*Cygwin*) é incorporada na arquitetura envolvendo *hosts* Windows, é previsto um possível aumento do tempo de resposta do SDI envolvendo esses *hosts*. O objetivo da avaliação, portanto, é verificar o impacto desse aumento no sistema.

Dependendo dos resultados obtidos, a implantação da portabilidade junto ao SDI poderia até mesmo ser descartada, ou então ser feita usando algum outro tipo de intermediador para realizar a comunicação, ou outro programa para funcionar como plataforma de execução dos *scripts*.

Como o sistema executa as operações em etapas, duas operações principais foram testadas separadamente: *(i)* aberturas de conexão e *(ii)* execução de *script*s objetivando identificar um possível "gargalo" no sistema.

O intuito da avaliação também foi de observar anomalias que possivelmente pudessem ocorrer. Anomalias tais como tempos de respostas absurdamente diferentes ou impossibilidade de execução de *scripts*.

5.2 AMBIENTE DE TESTE

Para o desenvolvimento do trabalho, foi usado o *computador 1* como servidor SDI e o *computador 2* como *host*.

- Computador 1
  - 1 GB RAM
  - Intel Celeron CPU 2.26 GHz
  - Ethernet: VIA Technologies 10/100

- Computador 2
  - 1,5 GB RAM
  - Intel Pentium 4 3,0 GHz
  - Ethernet: Realtek Semiconductor 10/100

- Switch 3COM 4200 Giga 24 ports
  - Cabeamento 100 Mbps Ethernet

5.3 METODOLOGIA

Os testes foram divididos em três etapas: *(i)* aberturas de conexão, *(ii)* execução de *script*s comumente usados pelo SDI e *(iii)* execução de *script*s em busca de anomalias.

Com o intuito de garantir as mesmas condições durante a realização dos testes nos clientes Linux e Windows, os mesmos testes foram realizados em um mesmo *host* com ambos os sistemas instalados, sendo que a cada bateria de testes os sistemas eram alternados. E é claro, manteve-se o mesmo servidor

SDI, para ambos os casos. Para cada um dos testes realizados, foram extraídos 25 resultados, e uma média aritmética desses valores foi definida como valor final. O número de 25 resultados foi escolhido de maneira arbitrária, apenas com a intenção de buscar um resultado mais preciso, minimizando a interferência externa (ruídos na rede, troca de contexto do S.O etc) no momento da aquisição do resultado.

5.3.1 Testes de Abertura de Conexão

Para a realização desse teste, um simples *script* foi desenvolvido. Ele captura a hora atual de acordo com o relógio do sistema, executa a operação de abertura e fechamento de um túnel SSH e captura novamente a hora atual do sistema. Subtraindo o segundo tempo pelo primeiro, será obtido o exato tempo de estabelecimento da comunicação.

A execução desse *script* é realizada no servidor, ou seja, é a partir dele que as conexões são abertas e o tempo é contado. Tempo esse que inclui o processamento local, atraso na rede, tempo de abertura/fechamento de conexão englobando o funcionamento do protocolo SSH em cima do TCP.

5.3.2 Testes de Execução de *Scripts* do SDI.

No diretório "*cmds*" dentro do SDI existe um arquivo específico para cada *host*, onde podem ser inseridos comandos para serem executados. Assim é possível criar *scripts* aleatórios dentro desses arquivos para que o próprio SDI execute-os imediatamente nos *hosts* e contar o tempo da operação. Neste caso, os *scripts* executados foram desenvolvidos para uso real no SDI implantado no LPRM.

Como exemplos, serão analisados três *scripts* comumente usados no SDI: *rammermory*, *so* e *uptime*. "***Rammemory"*** exibe o total de memória existente no *host*, "***so***" exibe qual o sistema operacional está sendo usado pelo *host* e finalmente, "***uptime***" conta e exibe o tempo o qual o *host* encontra-se ativo.

5.3.3 Testes de Execução de *Scripts* para a Busca de Anomalias.

A execução desses testes se deu da mesma forma que os testes da seção anterior, exceto pelos *scripts* utilizados. Como exemplos serão analisados dois *scripts* que seguem uma metodologia não comumente usada pelo SDI usado no LPRM: O comando ***netstat*** puramente aplicado, e o *script* "***foo***", que cria uma pasta, copia um arquivo para esta pasta e apaga a pasta criada, 150 vezes. Esse último *script* não é nada convencional, mas foi determinante para a observação de uma importante anomalia apresentada por essa proposta.

5.4 RESULTADOS

A avaliação dos resultados obtidos tem a intenção de determinar a decisão de realmente usar o *Cygwin* ou pensar em alguma outra alternativa para realizar a portabilidade.

5.4.1 Comunicação

Os tempos obtidos foram bastante destoantes. A média obtida para abertura de túneis SSH Linux-Linux, foi de aproximadamente 0,20 segundos, enquanto com as conexões Linux-Windows, a média foi de aproximadamente 5,37 segundos.

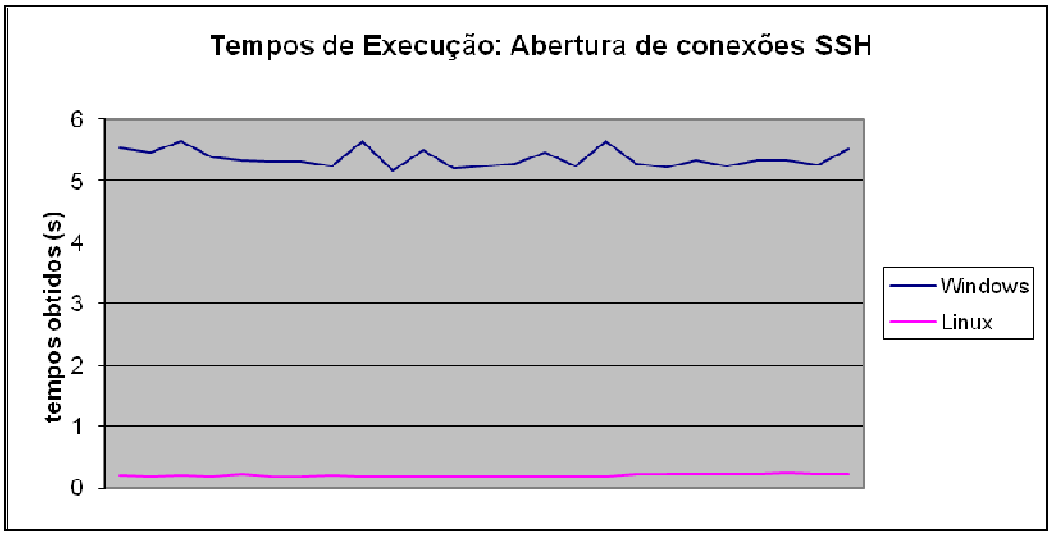

Figura 5-1: Gráfico dos tempos de resposta para as aberturas de conexão SSH

O gráfico ilustrado na figura 5-1 mostra os tempos obtidos (eixo Y) no decorrer das 25 execuções do *script* (eixo X). Essa diferença de tempo é justificada pelo fato de haver mais uma camada para a realização da conexão. Além disso, o *Cygwin* possui uma DLL que age como uma API de emulação do Linux, ou seja, a autenticação do usuário e iniciação do shell, antes de acessar o SO, precisam ser traduzidos pela DLL.

Ainda existe a questão da criptografia e des*cript*ografia das senhas/chaves, que também é feita em cima da DLL. As operações envolvendo codificação de dados são muito custosas, mesmo quando realizadas diretamente no SO.

5.4.2 *Script*s do SDI

O primeiro *script* a ser analisado é o ***rammemory***, que obteve uma média para o tempo total de execução de 0,905 segundos no *host* Linux. Para o *host* Windows, a média obtida foi de 0,958.

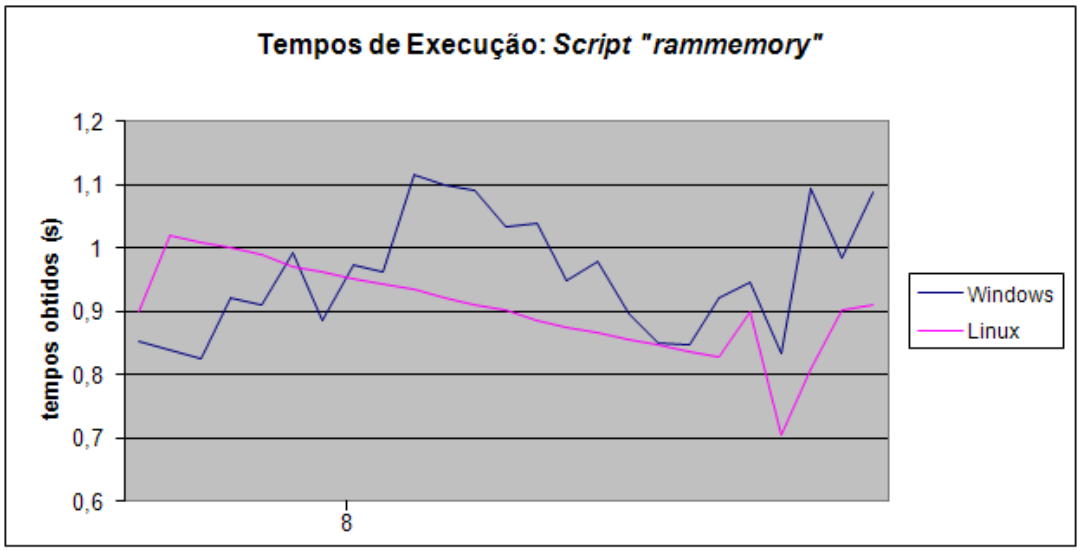

Figura 5-2: Gráfico dos tempos de resposta para a execução do *script rammemory*

Pode-se observar no gráfico da figura 5-2, que para os valores iniciais, os tempos de reposta do *host* Windows, são até mesmo mais baixos que os tempos apresentados pelo *host* Linux, mas a partir da oitava medida, o Linux passa a apresentar sempre um tempo de resposta menor.

O segundo *script* analisado (***so***) obteve uma diferença pouco maior entre os sistemas utilizados. A média para os tempos obtidos no *host* Windows foi de 0,699 segundos, enquanto para o *host* Linux, a média foi de 0,414 segundos. No gráfico ilustrado na figura 5-3, pode-se reparar que todos os valores obtidos no Linux são menores que os abtidos no windows.

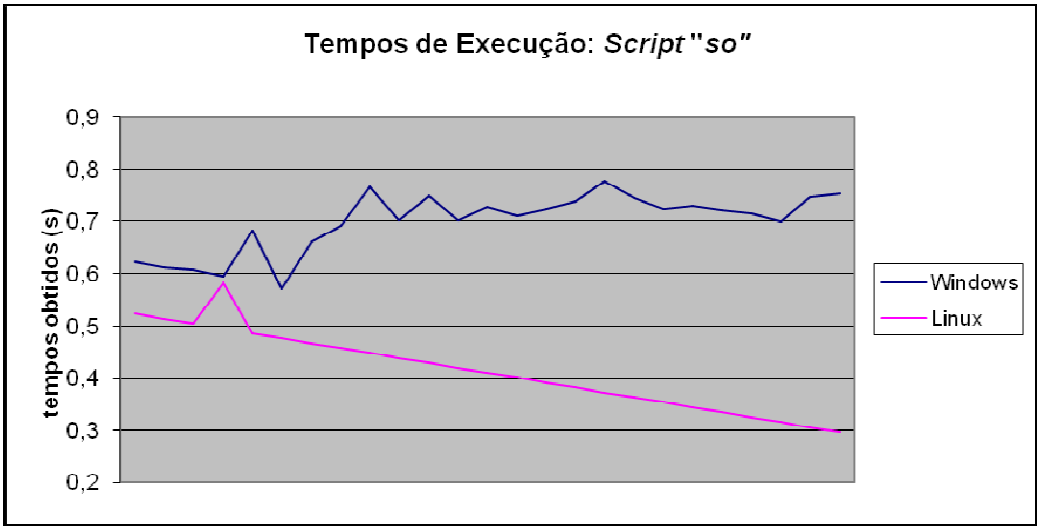

Figura 5-3: Gráfico dos tempos de resposta para a execução do *script so*

O terceiro e último *script* (***uptime***) foi o que apresentou a menor media dos tempos de resposta em ambos os sistemas. Para o *host* Windows, a média foi de 0,488 segundos, enquanto para o *host* Linux foi de 0,173 segundos. Pode-se notar que a linha referente ao Linux no gráfico 5-4 está sempre abaixo da linha correspondente ao Windows, mesmo que por pequena distância.

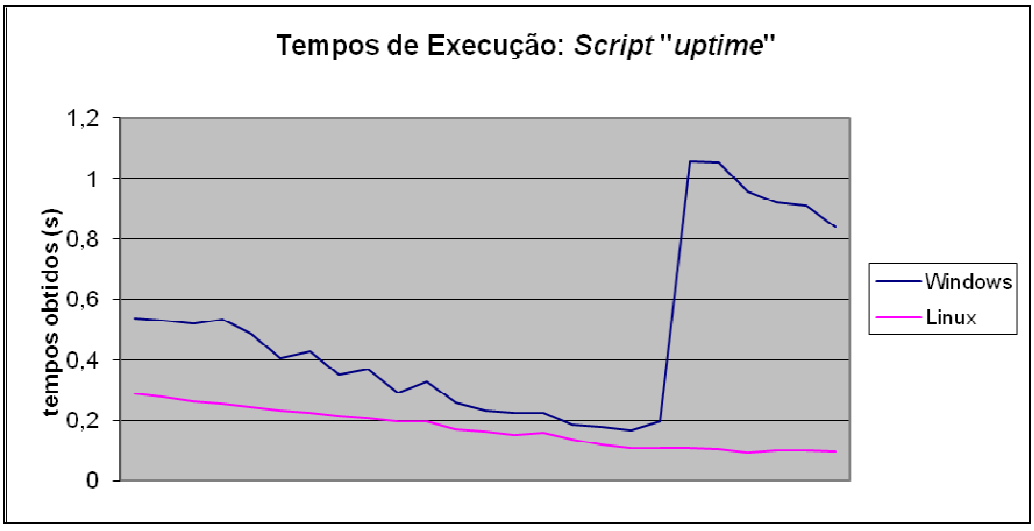

Figura 5-4: Gráfico dos tempos de resposta para a execução do *script uptime*

Por fim, pode-se observar nos últimos três gráficos, uma linha rosa quase que descentente em todos os casos. Com isso conclui-se que para cada rodada, as execuções dos *scripts* no Linux quase sempre eram mais rápidas do que nas rodadas anteriores.

Uma explicação possível para esse comportamento, é que o Linux mantém no *buffer cache*, os blocos de arquivos recentemente abertos. Como gradativamente esses blocos estavam sendo mais acessados, acabavam se tornando prioritários e consequentemente mantidos em memória principal, o que tornava seu acesso cada vez mais rápido.

5.4.3 *Script*s Aleatórios para Detecção de Anomalias.

Os *script*s analisados aqui foram selecionados em uma busca por situações inusitadas que pudessem ocorrer devido à implementação da nova funcionalidade de portabilidade no SDI.

O primeiro *script* (***foo.sh***) é basicamente formado por uma estrutura de repetição, que executa uma série de comandos, muitas vezes. O *script* testado apresentou uma média dos tempos de reposta igual a 5,679 segundos para o *host* Linux, enquanto essa mesma média para o *host* Windows foi de 13,756 segundos.

Repare no gráfico, que a diferença entre os tempos obtidos é constante e muito alta. Isso ocorre, pois o *Cygwin* precisa emular os comandos, por meio da DLL, para que o somente depois dessa operação, o *hardware* execute o que foi pedido. Quando o comando é dado uma única vez, essa operação aparentemente não é custosa, mas quando executada 450 vezes (são 3 comandos executados 150 vezes pela estrutura de repetição do *script*) a diferença fica bem visível como no gráfico da figura 5-5.

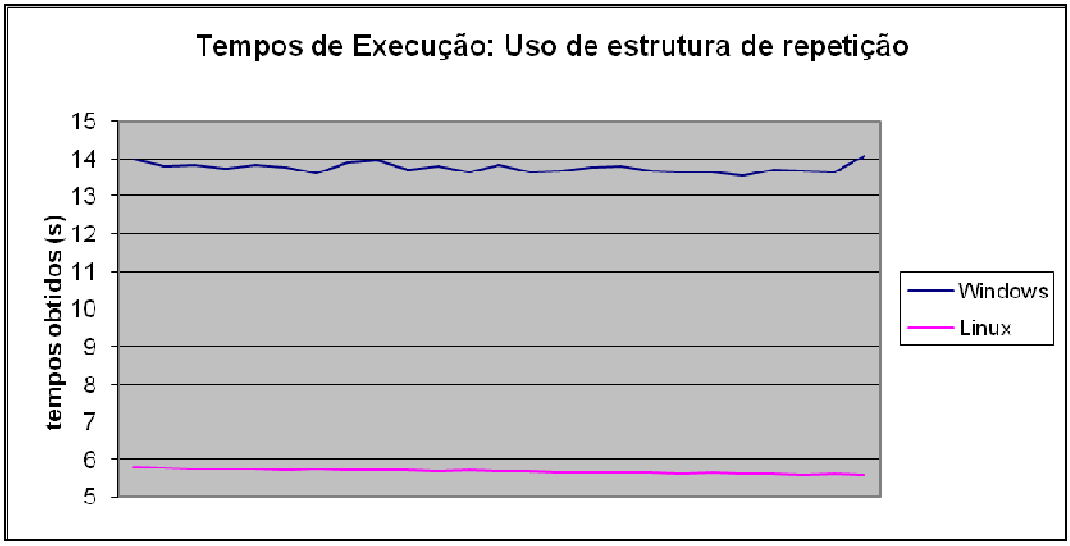

Figura 5-5: Gráfico dos tempos de resposta para a execução de um *script* que apresenta estruturas de repetição.

A segunda opção de *script* envolveu usar o comando *netstat*, que como dito por Nemeth (1995) - imprime na tela variadas informações sobre a rede e tem quatro funcionalidades mais comuns: exibir *status* das conexões de rede, inspecionar as interfaces, examinar a tabela de rota, exibir estatísticas operacionais de vários protocolos de rede.

O *script* testado apresentou uma média dos tempos de reposta igual a 3,472 segundos para o *host* Linux, enquanto essa mesma média para o *host* Windows foi de 0,422 segundos.

Dessa vez, os papéis se inverteram, a diferença agora está a favor do *host* Windows, como é visto no gráfico da figura 5-6. Isso ocorre, porque o *netstat* verifica os estados da rede junto às estruturas de dados do kernel. No caso do Linux, todos os estados dos *sockets* existentes na máquina são analisados, já no Windows, o limite é bem maior, uma vez que apenas os estados dos *sockets* ligados ao *Cygwin* são analisados. Ou seja, a resposta do comando *netstat* no Windows mostrou-se não confiável.

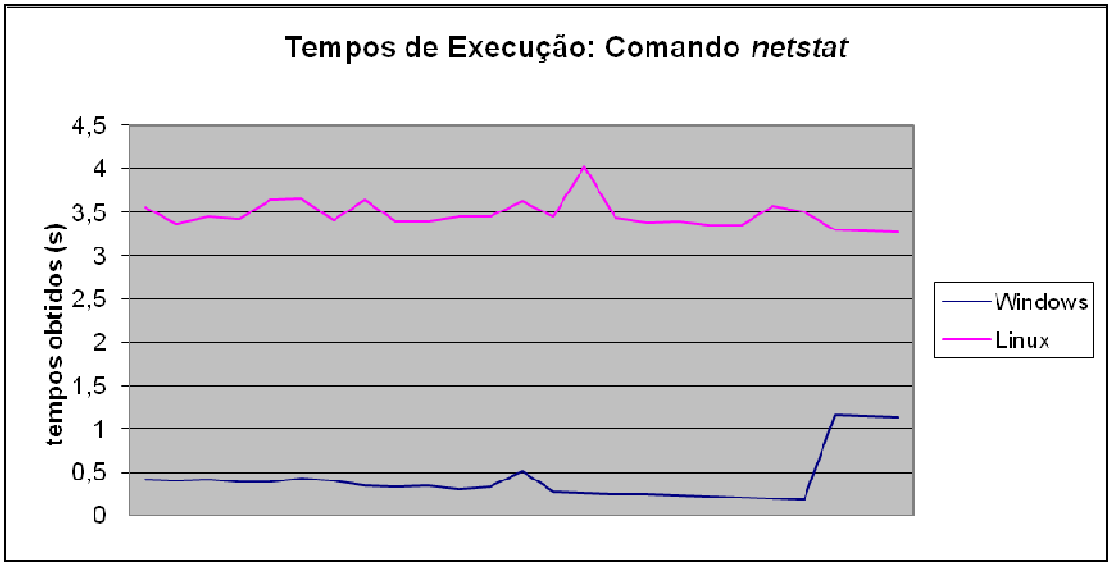

Figura 5-6: Gráfico dos tempos de resposta para a execução do comando *netstat*

5.5 CONCLUSÕES

Quando o SDI é lançado, o mesmo mantém a sua execução em *background*, e o usuário pode continuar a executar suas tarefas normalmente. Dessa forma, os túneis são abertos em um instante inicial de uma maneira transparente para o usuário, assim como *script*s usados pelo SDI para a coleta de informação ou correção de falhas.

A diferença do tempo de resposta dos *script*s entre *hosts* Linux e Windows é irrisória em contra partida à vantagem de um maior número de computadores agora poderem ser monitorados pelo SDI.

Tendo isso em vista, concluiu-se que é uma boa opção usar o *Cygwin* como pilar da portabilidade aplicada ao sistema. Mesmo que a diferença dos tempos de resposta para a abertura dos túneis seja considerável, ela ocorre apenas uma vez no inicio da execução do sistema. Além disso, a diferença nos tempos de respostas dos *scripts* foi bastante satisfatória, pois em virtude de serem tão pequenos, os dados sempre estarão atuais na interface do SDI.

Outra conclusão retirada dessa avaliação, é que nem tudo que é feito para Linux pode-se fazer para Windows esperando um mesmo resultado, como o caso do *netstat*.

Finalmente, *scripts* que apresentem estruturas de repetição com muitas iterações, não são recomendados para serem utilizados em *hosts* Windows, uma vez que essa operação se mostrou tão custosa, mesmo sendo realizada localmente (funcionando sobre o *Cygwin*).

6 CONCLUSÃO

Primeiramente o SDI foi estudado de forma a obter conhecimento suficiente para que extensões pudessem ser realizadas. Após a fase de estudos, deu-se inicio à fase de instalação do sistema. Durante essa segunda fase, foram realizados testes com *scripts* já presentes no SDI e posteriormente, *scripts* de autoria própria foram desenvolvidos, visando uma adaptação maior ao sistema.

Após a adaptação ao SDI ser concluída, e o entendimento do funcionamento ficar completo, deu-se início à fase de estudos para a criação de extensões. A primeira extensão realizada foi a criação da portabilidade, logo após foi feita a integração do SDI com o Nagios, e na sequência foi concebida a idéia de usar o SDI para além de diagnosticar problemas, também resolvê-los.

A partir deste trabalho, concluiu-se que é de suma importância a implantação de uma ferramenta de monitoramento que identifique situações de risco, e que o administrador seja avisado sobre essa situação o mais rápido possível. Com a implantação do SDI, foi possível verificar essa importância no contexto de laboratórios com diferentes tipos de máquinas (ambiente híbrido). Após a implantação do sistema, e dos testes realizados, pôde-se comprovar que o SDI tratava-se realmente de uma ferramenta extensível.

Os objetivos que foram propostos no capítulo 1 foram alcançados com êxito, tendo em vista que hoje o SDI está instalado no LPRM e funciona entre sistemas Windows ou Linux, usando inclusive *scripts* disponibilizados pelo Nagios, que foi a ferramenta escolhida para ser integrada ao SDI. Hoje o SDI também monitora os processos atuantes nos *hosts*, além de controlar uso de recursos físicos das máquinas como memória e processamento.

Também foi concluído que os resultados obtidos na avaliação (feita no capítulo cinco) foram satisfatórios, tendo em vista que os prós se mostraram bem maiores que os contras. Mesmo que existam algumas desvantagens na

aplicação da portabilidade no sistema, como por exemplo, o caso do aparecimento de algumas anomalias (sessão 5.4.3), ainda assim ela deve ser realizada, considerando que somente o fato de que o raio de atuação do sistema poderia englobar todas as máquinas da rede já seria uma grande vantagem para o administrador.

Uma contribuição importante deste trabalho foi a disponibilização dos códigos fonte que fazem a interoperabilidade com o Nagios, e também a criação de um *howto*, disponível no *site* dos desenvolvedores, descrevendo os passos da realização da portabilidade. Também estão disponíveis os *scripts* desenvolvidos com a finalidade de resolução/prevenção de problemas. Dessa forma os resultados obtidos neste trabalho podem ser utilizados no contexto de outros ambientes em rede que tenham demandas similares ao do LPRM.

6.1 TRABALHOS FUTUROS

O SDI mostrou ter um grande potencial para aplicações futuras, tendo em vista que o grau de liberdade apresentado é muito alto, e com isso, o sistema pode caminhar em muitas direções. Tudo depende da necessidade de quem utiliza.

6.1.1 Geração de Gráficos e Integração com Outras Ferramentas

Para a visualização dos dados, uma melhor opção, é utilizar gráficos que mostrem os resultados de uma maneira intuitiva. Hoje o SDI oferece um *output* que por padrão é texto, e um trabalho futuro é fazer com que a exibição dos dados coletados seja feita graficamente.

O Cacti é uma ferramenta que apresenta uma exibição gráfica muito boa, e uma tarefa futura, seria portanto, realizar uma integração com o Cacti, para que ele mostre graficamente os dados coletados pelo SDI, ou até mesmo construir um novo módulo no SDI que desenvolva essa função.

6.1.2 Criação de uma Interface Gráfica para o Sistema

O SDI exibe as informações em uma interface *Web*, mas toda a configuração, processo de criação de *script*s, lançamento e encerramento do sistema é feita via comandos texto. A criação de uma interface gráfica facilitaria essas ações.

A idéia da criação de uma interface gráfica, é que por trás dela, existisse um agente que teria o papel de dinamizar e automatizar alguns passos, como por exemplo, a criação do *parser object*. Além disso, a interface visaria aumentar o número de interessados na ferramenta, inclusive usuários finais que desejassem monitorar pequenas empresas, ou sua própria casa.

6.1.3 Usar o Protocolo *Jabber* para Comunicação

Neste momento, foi decidido que não é uma boa alternativa trocar o modelo de comunicação entre cada cliente-servidor, pois isso implicaria em uma mudança drástica do comportamento do sistema, como foi citado na sessão 4.2.1. Porém, para o futuro, essa seria uma ótima opção, tendo em vista que a nova versão do SDI, que atualmente está em fase de testes, irá oferecer um *plugin* que permitirá o administrador definir qual o padrão de comunicação que ele deseja estabelecer, permitindo tanto que a conexão seja feita via SSH, como outra forma de comunicação.

Mesmo que seja necessário mudar o funcionamento normal do programa, essa é uma alternativa a se pensar no futuro, pois é um meio rápido de comunicação, que possivelmente tornaria o SDI mais eficiente, em se tratando de comunicação entre diferentes sistemas, pois neste caso, não existiria mais o papel de intermediador feito pelo *Cygwin.* Agora as conexões seriam feitas diretamente entre os sistemas operacionais.

7 REFERÊNCIAS

1. ANDRADE, A. H. **Nagios como solução de monitoramento de rede**. 2006. 69 f. Trabalho de graduação (Graduação em Ciência da Computação) - Instituto de informática, Universidade Federal de Lavras, Lavras, 2006.
2. BARRETT, D. & SILVERMAN, R. **SSH, The Secure Shell: The Definitive Guide.** Sebastopol: O'Reilly, 2001
3. BARTH, W. **Nagios: System and Network Monitoring**. San Francisco: NO STARCH PRESS, 2006.
4. BLACK L. T. **Comparação de Ferramentas de Gerenciamento de Redes**. 2008. 64 f. Especialização (Trabalho de Conclusão apresentado para obtenção do grau de Especialista) – Instituto de informática, UFRGS, Rio Grande do Sul, 2008
5. FREITAS, S. C. & MEFFE, C. (2008). **A Produção Compartilhada de conhecimento no Portal do *Software* Público Brasileiro**. Disponível em: <http://www.ip.pbh.gov.br/ANO10_N2_PDF/producao_compartilhada_conhecimento.pdf>. Acessado em 04 de out. de 2009.
6. HARPERS, P., KROEGER, D., RIES, T. (2008) **SSH Inside Out**. Disponível em: <http://www.Linuxdays.lu/downloads/Linuxdays2008/0~ssh-inside-out-all-v0-5.pdf>. Acesso em 16 de nov. de 2009.
7. KOCH M. **Uma Proposta de Solução de Gerenciamento de Contabilização Utilizando Cacti e Nagios**. 2008. 36 f. Especialização (Trabalho de Conclusão para obtenção do grau de Especialista) – Instituto de informática, UFRGS, Rio Grande do Sul, 2008
8. KREUCH J. ***Software* de Inventário de *software* de Equipamento de Rede Utilizando *Session Message Block***. 2007. 46 f. Trabalho de graduação (Graduação em Ciência da Computação) - Instituto de informática, Universidade Regional de Blumenal, Blumenal, 2007.
9. KUROSE, F. J., ROSS W. K. **Redes de Computadores e a Internet: Uma abordagem top-down**. São Paulo: Addison Wesley, 2006.

10. LEME S. P. L, (2006). Apresentação. **Nagios.** Disponível em: <http://alesauer.googlepages.com/NAGIOS-ApresentaoSauer.pdf>. Acessado em: 14 de nov. 2009.
11. LIGOCKI, P. N. **Uma Ferramenta de Monitoramento de Redes Usando Sistemas Gerenciadores de Streams de Dados**. 2008. 83 f. Dissertação (Mestrado em Informática) – Ciências Exatas, UFPR, Curitiba, 2008.
12. LIGOCKI, P. N., HARA, S. C. (2007) **Uma Ferramenta de Monitoramento de Redes usando Sistemas Gerenciadores de Streams de Dados**. Disponível em: <http://www.inf.ufpr.br/carmem/pub/wgrs07.pdf>. Acessado em 02 de dez. 2009.
13. MOCERI, P. (2004) **SNMP and Beyond: A Survey of Network Performance Monitoring Tools**. Disponível em: <http://www.cs.wustl.edu/~jain/cse567-06/ftp/net_traffic_monitors2.pdf> Acessado em 02 de dez. 2009.
14. MORIMOTO, E. C. (2002). ***Hardware*** **Manual Completo**. 3ª Edição. Versão online. Disponível em <http://www.gdhpress.com.br/hmc/>. Acesso em: 07 de nov. 2009.
15. NEGUS, C. **Linux: a bíblia. *Boot* up Ubuntu, Fedora Knoppix, Debian, Suse e outras 11 distribuições.** Rio de Janeiro: Alta Books, 2006.
16. NEMETH, E., SNYDER, G., SEEBASS, S., HEIN, R. **Unix System Administration Handbook**. Second Edition. New Jersey: Prentice Hall PTR, 1995
17. **OCS inventory next generation: Installation and Administration Guide**. Version 1.9. Disponível em: <http://www.ocsinventory-ng.org/index.php?page=1-01>. Acesso em: 19 de nov. 2009.
18. RIBAS, C. B., PASQUALIN G. D., RUOSO, K. V., et al (2009) **SDI – Sistema de Diagnóstico Instantâneo**. Disponível em: <http://sdi.sourceforge.net/files/wsl2009.pdf>. Acessado em 03 de set. 2009.
19. SHIGEOKA, I. **Instant Messaging in JAVA: The jabber Protocols**. Greenwich: Manning Publications, 2002.

20. SILVEIRA, F. C. M. R e ROSA, S. G. M. (2009). **PROJETO PARANÁ DIGITAL: A Tecnologia favorecendo o professor em sala de aula**. Disponível em: <http://www.pg.utfpr.edu.br/sinect/anais/artigos/11%20TICnoensinoaprendizagemdecienciaetecnologia/TICnoensinoaprendizagemdecienciaetecnologia_artigo14.pdf>. Acessado em 29 de set. 2009.
21. URREA, L. S., JIMINÉZ, V. D. e GONZÁLEZ, G. J. **Manual de Usuario de Pandora FMS: Versión PDF**. 1ª Edición. Madrid: Artica ST, 2009
22. VINSCHEN, C., FAYLOR, C., DELORIE, D. J., HUMBLET, P. AND NOER, G. (2001). ***Cygwin* User's Guide**. Disponível em: <http://www.*Cygwin*.com/*Cygwin*-ug-net/*Cygwin*-ug-net.html>. Acessado em: 12 de out. 2009.
23. ZANIKOLAS, S., SAKELLARIOU, R. **A Taxonomy of Grid Monitoring Systems.** Amsterdam: Elsevier, 2005.